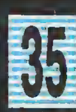
BIBLIOTECA BASICA INFORMATICA

UNIX  el futuro de los sistemas operativos

INGELEK

BIBLIOTECA BASICA INFORMATICA

UNIX 35 el futuro de los sistemas operativos

INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Monsalve.

Redactor técnico:
Alfredo B. G.ª Pérez

Colaboradores:
Casimiro Zaragoza

Diseño:
Bravo/Lofish.

Dibujos:
José Ochoa.

ISBN del tomo: 84-85831-81-0
ISBN de la obra: 84-85831-31-4
Fotocomposición Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-23.046-1986
Precio en Canarias, Ceuta y Melilla: 380 pts.



INDICE

PROLOGO

UNIX son las siglas de identificación de uno de los mejores sistemas operativos de aplicación general que se encuentran en el mercado de micro y miniordenadores en este momento. Y hemos dicho uno de los mejores por no decir el mejor. Nos explicaremos: UNIX es, ante todo, un sistema operativo de uso general es decir, que posee herramientas y utilidades capaces de efectuar aplicaciones de gestión, control industrial, comunicaciones, cálculo científico, diseño gráfico, etc. UNIX no está limitado a una aplicación concreta.

Uno de los objetivos fundamentales que se propusieron los creadores de UNIX fue su portabilidad y facilidad de implementación en diversos tipos de hardware. Esto se consiguió mediante la creación de un lenguaje de programación denominado C, y la escritura del núcleo del sistema UNIX en dicho lenguaje; con lo que únicamente se debe poseer un compilador de C en la máquina donde se desea implementar el sistema operativo UNIX. Los controladores de dispositivos (discos, cintas, terminales, impresoras, etc.) estaban escritos en código máquina o bien en ensamblador, con lo que se debían reescribir para cada tipo de hardware. Hoy en día casi todos están escritos en lenguaje C y para numerosos tipos de hardware que pueda soportar UNIX, con lo que se reafirma su portabilidad de una máquina a otra.

El lenguaje C está íntimamente ligado al sistema operativo UNIX y prácticamente no se pueden entender el uno sin el otro. Debido a que dicho lenguaje posee una sintaxis de alto nivel, tiene potencia para manejar el sistema a muy bajos niveles. Al estar

profundamente interrelacionado con todo UNIX se consiguió un sistema operativo con potencia para poder desarrollar cualquier tipo de utilidad. UNIX posee gran cantidad de complejos comandos y programas, pero más que nada ofrece unas posibilidades de programación, control de procesos, multitarea y control de flujos, que lo hacen ideal para cualquier tipo de aplicación, no estando limitado, como muchos sistemas operativos, a gestionar un tipo concreto de aplicaciones.

Veamos dos aspectos prácticamente contradictorios de UNIX: por un lado, es un sistema operativo de muy pocos conceptos básicos, muy claros y bastante fáciles de entender, que una vez dominados nos ofrecen unas posibilidades inmensas de trabajo; por otro, se tiene la impresión de que es un sistema operativo demasiado complejo y que **"falla"** en algunas ocasiones. Efectivamente, es un sistema complejo, como dijimos antes, pero en absoluto con fallos monumentales de cara al usuario. Han existido y existen numerosas versiones de UNIX, que van mejorando con el tiempo; nunca se podrá decir de UNIX, ni de ningún otro sistema operativo, que está creada la versión definitiva e inmejorable del mismo. Ahora bien, los fallos de UNIX son responsabilidad total del usuario y de la falta de conocimiento general que existe sobre este sistema operativo. Sí se puede decir que UNIX nunca pide confirmación para ejecutar el comando que se le ha enviado y, por este motivo, se pueden producir casos de alteraciones importantes de la información, pero todo ello es debido a la falta de preparación de los usuarios. Quizá se le pueda pedir a UNIX algo más de control en estos aspectos, pero no se puede desvirtuar un sistema como UNIX por un detalle de ese tipo. También carece de una buena documentación de cara al usuario, e incluso los manuales originales de UNIX son demasiado extensos y poco claros (crípticos).

UNIX posee una extensa biblioteca de programas especiales para muy diversas tareas y cada vez existen más en el mercado. Es un sistema que tiene sentadas las bases de su estructura y funcionamiento. Se irá mejorando, creemos, con el incremento de la potencia del hardware que incorporen las máquinas u ordenadores. Quizá la mayor tarea que le queda al UNIX sea el aprendizaje de los usuarios finales del mismo, e incluso del personal técnico que trabaja en entornos UNIX. Existe un numeroso conglomerado de libros y publicaciones sobre el tema, pero muchos no hacen más que contribuir a la confusión general. Nuestra opinión como usuarios de UNIX y, aún hoy, estudiosos del mismo, es que se deben recalcar mucho las ideas y estructuras básicas de UNIX durante el tiempo que haga falta, y no continuar con el resto del sistema hasta el dominio total de estos conceptos. Una vez que se tenga la confianza para poder seguir adelante, continuaremos navegando por la extensa carreta del UNIX, acudiendo siempre, y en último término, a los manuales originales.

Por todo esto, el libro que presentamos a continuación no pretende ser ni una sustitución de los manuales, ni una pequeña iniciación al sistema. Pretendemos recalcar y poner ejemplos de los conceptos básicos de UNIX de tal forma que el usuario pueda llegar a adquirir la confianza necesaria para continuar el estudio en otros niveles. La claridad y a la vez complejidad de UNIX y del lenguaje C nos han llevado a la idea de crear una obra de UNIX que, a la vez, sirva como introducción al sistema y profundice en numerosas partes que, si no se tratan desde un principio, pueden llegar después a ser totalmente oscuras para el usuario final. Cuando el usuario continúa adelante con el sistema sin esa fuerte base en los conceptos UNIX llegará a decir que el sistema "falla". Debemos hacer notar que sería muy conveniente compaginar la lectura del presente libro con el de Lenguaje C (número 29 de la Biblioteca Básica Informática), pues, como dijimos anteriormente, no se pueden entender el uno sin el otro.

El libro comienza con una reseña histórica del UNIX. A continuación explicaremos lo que se va a encontrar el usuario cuando entre en el sistema. Seguiremos con el desarrollo de los conceptos básicos sobre ficheros UNIX y los comandos para manejarlos, que es el apartado más extenso del libro. Más tarde nos adentraremos en un comando básico, el editor de ficheros **(ed).** Avanzaremos en el estudio del resto de los conceptos fundamentales de la Shell del sistema, finalizando con unas nociones sobre la estructura interna del núcleo del sistema. Añadiremos una completa tabla de comandos UNIX (exceptuando aquellos que sobrepasen el nivel de un usuario medio), con todas sus opciones posibles; esta tabla servirá de guía para el usuario una vez dominados el resto de los capítulos del libro.

Como todo libro de desarrollo de un sistema operativo conviene leerlo más de una vez, por la sencilla razón de que la complejidad de un sistema operativo, no solamente de UNIX, requiere el manejo de tantos conceptos y definiciones que es imposible seguir un orden totalmente coherente en el desarrollo de los temas. Así podemos encontrar al principio del libro conceptos que son ampliamente desarrollados posteriormente y que, en una segunda lectura, nos darían una visión mucho más completa y clara del tema tratado.

Esperamos con esto poder sentar las bases que precisa el mero interesado o el buen usuario y posterior programador en UNIX. No pretendemos hacer una obra extensa y compleja, que no llevaría más que al olvido posterior de todo lo tratado, sino una potente introducción a este sistema operativo. Esperamos haberlo conseguido.

CAPITULO I

DESARROLLO HISTORICO DEL UNIX

Necesidad del UNIX

EL UNIX es un Sistema Operativo que se desarrolló a principio de la década de los setenta por el personal de la "BELL TELEPHONE LABORATORIES" (NEW JERSEY-USA), pertenecientes a la compañía AT&T.

Estos laboratorios se establecieron en 1925 y se pueden considerar como uno de los más importantes grupos de investigación del mundo. En definitiva se dedican a realizar investigaciones dentro del mundo científico, como puden ser las comunicaciones; las matemáticas, la física y, de manera especial, la informática (diseño asistido por ordenador, técnicas de compilación, redes, etc.).

Durante el transcurso de 1969, el Departamento de Investigación Científica de Computadores de los Laboratorios Bell utilizó un ordenador 645 de General Electric con un sistema operativo denominado "MULTICS". Multics se puede considerar como uno de los primeros sistemas interactivos **multiusuario,** es decir, que permite la utilización simultánea del computador por varios usuarios y que además, al ser interactivo, ofrece una respuesta casi inmediata a la petición de una determinada orden.

Con anterioridad el método de trabajo era sustancialmente distinto, puesto que los sistemas operativos implementados funcionaban por **lotes.** El usuario mecanografiaba las órdenes destinadas al computador en fichas perforadas, éste las leía y algún tiempo después (que podía ser varios minutos o varias horas) se obtenían los resultados en papel impreso. Dada la lentidud del

proceso, el Multics supuso un importante avance; sin embargo, este sistema operativo carecía de características importantes para los programadores.

En 1969 Ken Thompson y Dennis Ritchie crearon un sistema operativo con el fin de paliar algunos problemas de complejidad que se derivaban de la utilización de grandes ordenadores. A este sistema se le llamó UNIX y supuso la simplificación del diálogo entre el hombre y la máquina

Thompson y Ritchie consideraron que la mayor parte del desarrollo de programas (más del 50%) se empleaba en crear ficheros fuente y compilarlos. Para tales tareas no es necesaria la utilización de una gran máquina, sino que basta con un ordenador pequeño que disponga de gran memoria en periféricos donde almacenar los ficheros.

Aspecto importante a considerar es que el programador disponga de un método sencillo para la creación, actualización y manejo de los ficheros; ésta es una de las características esenciales del UNIX: un sistema de ficheros con estructura jerárquica en forma de árbol Esta herramienta permite la creación de un fichero sin necesidad de definir parámetros del mismo (extensión, formato, etc.); se puede localizar, además, de forma automática cualquier fichero en cualquier directorio del sistema.

Una vez que se ha creado el fichero fuente es necesario compilarlo, y para tal tarea tampoco se requiere un gran ordenador, que únicamente será necesario a la hora de ejecutar grandes programas. Se desarrollaron, por tanto, en Bells Labs una serie de programas de utilidad para facilitar las comunicaciones entre los grandes y pequeños ordenadores. Estas utilidades forman parte del sistema UNIX y son accesibles por el usuario. Estos programas permiten la traducción del código interno utilizado por los pequeños PDP-11 de Digital, al código EBCDIC usado por los grandes ordenadores IBM.

Concluyendo, con el UNIX se logró dotar a máquinas pequeñas de un sistema interactivo comparable al de los grandes ordenadores de la época.

Desarrollo del sistema

La primera versión del UNIX se realizó durante los años 1967-1970 en los ordenadores PDP-7 y PDP-9 de Digital Equipment. Este versión, escrita en lenguaje Ensamblador, contenía los principales conceptos de sistemas como CTTS, TENEX, MULTICS. En un principio sólo permitía la utilización del ordenador en monousuario y el desarrollo del software fundamental (ensambladores,

compiladores, editores de texto y sistema) supuso cerca de cinco años-hombre de programación.

En 1971 se desarrolló la segunda versión del UNIX. Se utilizaron para ello el PDP-11/40 y el PDP-11/45. En esta nueva versión se introdujo como característica más importante la explotación del UNIX en modo multiprogramación.

Thompson desarrolló un lenguaje basado en el "BCPL" con la finalidad de que fuera transportable y lo denominó "Lenguaje B". En el mismo año 1971 Ritchie lo modificó y el nuevo resultado se llamó "Lenguaje C". El Lenguaje C se ha utilizado tanto para la escritura de sistemas y de software básico cómo para aplicaciones de usuario.

La re-escritura en el lenguaje C del Sistema UNIX se realizó en 1973 para algunas máquinas de la gama PDP-11, añadiéndose así nuevas funciones al sistema. En 1974 se escribió una versión simplificada del UNIX para el microordenador LSI-11/03.

Como mencionamos anteriormente, el software incluye muchos programas desarrollados en lenguaje C. Algunos se han incorporado como nuevas órdenes del sistema para usuarios o técnicos de sistemas, mientras que otros ejecuta funciones específicas. Por ejemplo, un programa actualiza una base de datos en un entorno de tiempo real, mientras que otro proporciona posibilidades de edición de textos. Los profesores y estudiantes universitarios han escrito numerosos programas C compatibles con el software de UNIX. La Western Electric vende licencias para el sistema UNIX con aproximadamente 250 programas.

Hasta el nacimiento de la versión VI de UNIX, su uso era interno y restringido a los laboratorios Bell. Estaba extendido por los distintos departamentos, que le fueron incorporando modificaciones de acuerdo con sus respectivas necesidades (MERT, PWD/UNIX).

En 1975 el sistema UNIX comienza a ser difundido. Su comercialización se limita al envío de una cinta magnética y un ejemplar de cada manual. Los laboratorios Bell no garantizaban por entonces ni la instalación, ni el mantenimiento del sistema. El **PWD** ("Programmer's Workbech") se desarrolló a la vez que la versión VII de UNIX para un equipo diferente al utilizado por Thompson y Ritchie. Se utilizó para centros de cálculo donde el desarrollo se hacía con PWD/UNIX en miniordenadores PDP-11 y la ejecución se realizaba en sistemas IBM 370 y UNIVAC 1100. Las innovaciones que conlleva el PWD con respecto a la versión VI de UNIX son:

- modificaciones en las funciones de organización de usuarios en proyectos de gran tamaño;
- creación de utilidades para gestión de las fuentes ("SCCS") y el envío de trabajos a distancia.

En los laboratorios Bell el sistema PWD/UNIX se puso a disposición del servicio B.I.S.P., que posee una red de PDP-11/45 y PDP-11/70 conectados con IBM 370/168, XDS SIGMA5 y UNIVAC 1100 utilizados por gran cantidad de usuarios.

La versión VII supuso un cambio importante con respecto a la VI, introduciendo una serie de modificaciones que se pueden resumir en:

- superación de las limitaciones en cuanto a tamaño de ficheros;
- intento de portabilidad con el fin de transportar el UNIX a distintas máquinas;
- desarrollo de nuevas utilidades.

Hasta entonces UNIX sólo se había instalado en máquinas PDP-11; sin embargo, se llega a la conclusión de que es necesario transportarlo a otras máquinas. El primer transporte se realizó en el año 1977 sobre un ordenador INTERDATA 8/32. El hecho de que fuera necesaria la transportabilidad supuso llevar a cabo una serie de modificaciones:

- a nivel del lenguaje C se realizaron los cambios oportunos para que los programas se independizaran en la medida de lo posible de las características físicas de la máquina;
- a nivel del compilador se introdujeron técnicas que facilitaban la adaptación del mismo a distintas máquinas;
- a nivel del sistema se pretendió aislar las partes más dependientes de la arquitectura de la máquina del resto.

Debido al aumento de potencia y al abaratamiento del coste de los minicomputadores, el sistema operativo UNIX se ha hecho rápidamente popular. Estas máquinas se han utilizado inmediatamente para el control de experimentos de laboratorio, soporte de diseño asistido por computador, supervisión de redes de telecomunicaciones y ejecución de funciones comerciales. El desarrollo de un software que cumpliera estas aplicaciones específicas supuso a los programadores un nuevo reto y el sistema UNIX se ofrecía como una herramienta efectiva para conseguirlo. Hacia 1978 estaban funcionando en universidades, departamentos gubernamentales y en los laboratorios Bell más de 500 instalaciones del sistema UNIX.

Con las modificaciones descritas enteriormente surge en 1978 la versión séptima del UNIX (UNIX/V7) para los ordenadores PDP-11 UNIX/V32 y VAX 11/780. El UNIX Sistema III, que aparece en 1982, incluye además:

- utilidades del PWB/UNIX;
- una distribución para máquinas de la gama de Digital Equipment (del PDP-11/23 al VAX 11/780);
- mecanismos de comunicación entre procesos.

El sistema V, que surge en 1983, ofrece también la licencia de instalación, mantenimiento y actualizaciones del sistema. En la actualidad es importante destacar el sistema UNIX desarrollado por la Universidad de Berkeley (versión 4.1. BSD), que gestiona memoria virtual. Estos programas han sido adoptados por los laboratorios Bell y se incluyen en las nuevas versiones de UNIX. Una de estas importantes contribuciones es el editor de texto **vi.**

En definitiva, hay dos opciones posibles en la elección del UNIX como herramienta de trabajo:

- los sistemas que trabajan con el UNIX transportado a una máquina en particular. En este caso es necesario el pago de una licencia y de un canon a los laboratorios Bell;
- los sistemas que poseen los mismos servicios que el UNIX pero que han sido lo suficientemente modificados como para que se los considere ajenos a los laboratorios Bell.

Los programas UNIX

Los programas del sistema UNIX están funcionalmente clasificados de la siguiente forma:

- El núcleo, que planifica tareas y gestiona el almacenamiento de datos.
- La Shell es un programa que relaciona e interpreta las órdenes tecleadas por un usuario del sistema. Interpreta peticiones de usuarios, llama a programas de la memoria y los ejecuta al mismo tiempo o en unas series de canalizaciones llamadas "tuberías" (pipes).
- Los programas de utilidad ejecutan una variedad de subrutinas y unas funciones especiales de mantenimiento del sistema.

El sistema UNIX se puede complementar o modificar por todo aquel que disponga de licencia de acceso a los códigos fuente del mismo. Un elevado número de programadores han ido incorporando al sistema programas, incluyendo los de la Universidad de California. Estos programas, y los que irán apareciendo, incrementarán la amplia biblioteca de software existente en UNIX.

CAPITULO II

EL USUARIO FRENTE AL SISTEMA UNIX

Cuando se pretende acceder al sistema operativo UNIX lo primero que debemos hacer es presentarnos al mismo. Esto se consigue con un identificador que permita al sistema comprobar si se poseen los derechos propios de acceso, identificador que es asignado por el administrador del sistema. También es necesario disponer de un terminal del tipo conversacional conectado al sistema para poder efectuar esa tarea de presentación al sistema.

Estos procedimientos se denominan **login,** y pueden ser efectuados mediante el comando **login <nombre de usuario>** o bien automáticamente después del enganche y conexión del terminal al sistema. Explicaremos la utilización del teclado del terminal con sus caracteres especiales y simularemos una sesión de trabajo UNIX para ir profundizando y desarrollando algunos comandos básicos y fundamentales.

Terminales y su utilización

La comunicación entre el sistema operativo UNIX y el usuario se realiza fundamentalmente a través de un terminal o consola (entrada-salida). El sistema UNIX trabaja bajo el modo de transmisión **"full duplex",** es decir, los caracteres que se digitan en la consola son enviados al sistema y éste responde reenviándolos a la pantalla, con lo cual son visualizados por el operador. En casos concretos y especiales, como puede ser la digitación de una palabra

secreta de paso o acceso al sistema, este eco a pantalla es eliminado.

Muchos de los caracteres que se pueden digitar son directamente visualizables en pantalla, pero existen caracteres especiales que poseen otra interpretación y significado para el sistema. Uno de los más importantes es el carácter generado por la tecla de RETURN. Este carácter indica al sistema el fin de la línea de entrada, moviendo el cursor al principio de la siguiente línea. El carácter RETURN debe ser enviado al sistema antes de que éste interprete la secuencia de caracteres enviada previamente. Como podemos observar, es uno de los caracteres de **control** (caracteres invisibles), que producen una serie de efectos especiales en nuestro terminal. Este carácter RETURN está implementado directamente sobre el teclado de la mayoría de los terminales. Otros caracteres de control deben ser generados presionando una tecla especial denominada **CTRL, CTL o CTNL** y otro carácter, que normalmente suele ser una letra. El carácter RETURN es equivalente a presionar la tecla CTRL seguida de "m". Existe otra serie de caracteres de control que definimos a continuación:

- <CTRL> d ⟶ Indica al proceso que se ha finalizado la entrada de caracteres. No hay más caracteres en la entrada.
- <CTRL> h ⟶ Efecto de **backspace.** Se utiliza para la corrección de errores de tecleo.
- <CTRL> i ⟶ Tabulador. Avanza el cursor de la pantalla hasta el próximo tabulador definido (tab stop). En UNIX la distancia entre tabuladores es de 8 espacios.
- <DELETE> / <RUBOUT> / <CTRL> c ⟶ En muchos sistemas UNIX este carácter produce la finalización de un proceso sin esperar a su término:
- <BREAK> ⟶ Esta tecla, dependiendo de cómo esté definido el terminal, produce efectos similares a la DELETE o RUBOUT.

Una sesión de trabajo con UNIX

Vamos a simular una sesión de trabajo con el sistema y ver algunas de sus características. A continuación de cada comando se incluye una explicación del mismo (tenga en cuenta que esa parte no entra dentro del diálogo establecido con el sistema):


```
                                          Efectuamos la conexión al
                                          sistema mediante el en-
                                          cendido del terminal o la
                                          comunicación telefónica.
                                          El sistema le responderá
Sistema UNIX - PDP-8                      con el mensaje:
login: pruebas                            Nombre de usuario y RETN.
Password:                                 No se hace eco a pantalla
You have mail.                            Aviso de que hay correo.
$                                         Se puede digitar comandos
$                                         Se ha presionado RETN.
$ date                                    Se solicita fecha y hora.
Sat May 17 12:20:54 EDT 1986
$ who                                     Quien usa el sistema.
desarrollo     tty00    May  17 12:04
pruebas        tty02    May  17 09:25
pedro          tty03    May  17 11:00
procesos       tty05    May  17 10:30
$ mail                                    Se solicita el correo.
From pedro Sat May 17 11:02 EDT 1986
Llámame por teléfono mañana, por favor.
Gracias. Saludos.

                                          RETN siguiente mensaje
?
From director Sat May 17 08:33 EDT 1986
Déjame los informes en mi mesa.
No voy a estar en todo el día.

? d                                       - Se borra el mensaje
$                                         No hay más correo.
$ mail director                           Envio de correo a director
mañana recogeré los nuevos discos.
ctrl-d                                    Fin de correo a director
$ ctrl-d                                  Salida del sistema
login:
```

En las próximas secciones desarrollaremos esta sesión e incluiremos algunos otros comandos de utilización dentro del sistema.

Entrada al sistema (login)

Antes de continuar vamos a definir y explicar el concepto de usuario y grupo de usuarios del sistema. El sistema asocia a un usuario con su nombre identificador, es decir, no reconoce personas físicas sino, digamos, *razones sociales*. Por lo tanto, se pueden tener diferentes terminales del sistema trabajando para el mismo usuario. Podemos definir al usuario como la entidad lógica del sistema sobre la que se va a producir todo el trabajo del operador en una sesión con el mismo. Cualquier fichero, programa o documento creado dentro de una sesión pertenecerá al usuario en

el que se encuantra y todos los procesos serán ejecutados dentro de ese usuario al que se ha concectado el operador.

Un usuario pertenece a una agrupación de usuarios, el **grupo,** de manera que se pueden compartir con los usuarios de un mismo grupo los derechos de acceso a determinados ficheros comunes. Todo esto tiene interrelación con los permisos de acceso a ficheros que serán explicados en el capítulo correspondiente. Así pues, un usuario puede ser propietario de un fichero, miembro del mismo grupo que el propietario del fichero o incluso no tener nada que ver con él (resto de usuarios). La asignación de un usuario a un grupo se produce en el momento de la creación del mismo por el administrador del sistema y dentro del fichero /etc/passwd. Cada usuario posee un número identificador del mismo, que es asignado por el sistema en el momento de la creación del usuario. El número de usuario "0" corresponde al administrador del sistema y su nombre es **root.** Trabajando en este usuario se poseen los máximos privilegios del sistema y se pueden efectuar tareas de mantenimiento del mismo. Su nombre común de identificación es **super user.** Este súper-usuario se encargará de mantener la integridad del sistema de ficheros, restaurarlo en caso de caída del mismo o error de operación, mantener los usuarios que se vayan a definir y, en general, efectuar tareas de mantenimiento del sistema. Existen comandos que solamente puede efectuar el super usuario.

Trabajando dentro de cualquier usuario que no sea **root** se pueden adquirir los privilegios de este mediante el comando **su.**

```
$ date
Sat May 17 14:56:10 EDT 1986
$ date 1810                         orden de cambio de hora.
date: no permission                 no hay privilegio del sistema
Sat May 17 14:56:20 EDT 1986
$ su                                entrada en privilegio máximo
Password:                           Solicita la palabra  de paso
                                    del usuario root.
# <RETN>                            Prompt # de super usuario
# date 1920                         Solicitud de cambio de hora
Sat May 17 19:20:00 EDT 1986        Hora cambiada del sistema
# ctrl-d                            Vuelta a usuario normal
$                                   Prompt de usuario normal
```

Como vemos, la palabra clave del usuario **root** debe ser conocida por muy pocas personas, puesto que nos garantizará la integridad del sistema ante posibles operadores inexpertos o malintencionados que pudiesen entrar en súper-usuario. Cuando se tiene el privilegio de súper-usuario, el sistema no efectúa control

alguno sobre los procesos que se le ordenan ejecutar, con lo cual la responsabilidad del mismo es grande.

El sistema operativo posee una lista definida de usuarios (aparte de los de gestión y base del sistema), definidos por un nombre, una palabra clave de acceso y una serie de atributos internos. Para la entrada en el sistema es necesario el conocimiento del nombre y palabra de acceso al usuario

Supongamos que el terminal utilizado tiene la posibilidad de trabajar con caracteres alfabéticos en minúsculas y mayúsculas. Debemos comprobar que todos los switches del terminal están correctamente instalados para trabajar bajo sistema operativo UNIX (consultando el manual del terminal se puede obtener la configuración correcta). Entre éstos, los más importantes son la velocidad de transmisión (baudios), modo "full duplex", paridad y mayúsculas-minúsculas. Una vez conectado el terminal, nos aparecerá en el mismo el mensaje:

login:

En respuesta a este mensaje, el operador debe digitar el nombre del usuario donde se va a producir la sesión de trabajo y presionar la tecla RETURN. Si el usuario (que habrá definido el administrador del sistema) posee una palabra de acceso, el sistema la solicitará y el operador deberá digitarla correctamente, no produciendo el sistema eco en pantalla para preservar el secreto de dicha palabra.

La culminación del proceso de entrada será la presentación en pantalla de un "prompt" (carácter significativo de que se está bajo el control del sistema operativo) que normalmente es el carácter **"$"** o **"%"**, pero que puede ser cambiado o redefinido por el administrador del sistema mediante el comando apropiado. Este prompt es enviado por un programa denominado **"intérprete de comandos (command interpreter) o Shell"**, que es el interfase del usuario con el sistema operativo. Antes de la visualización del prompt se puede obtener la fecha del día, un mensaje de aviso de que tenemos correo pendiente o bien indicación del tipo de terminal que se está utilizando.

La digitación en el teclado del nombre de usuario debe ser efectuada en minúsculas, puesto que si se hace en mayúsculas, el sistema utilizará posteriormente ese mismo tipo de letra en todas las comunicaciones del usuario, entendiendo que el terminal carece de la posibilidad de trabajar en mayúsculas-minúsculas.

Existe un comando para el cambio de la palabra de paso o acceso a un usuario. Para ello se debe entrar (login) en el usuario deseado y digitar el comando **passwd:**

```
$ passwd
Old password:
New password:
Retype new password:
$
```

Una vez digitado el sistema solicitará la palabra actual de paso del usuario, que deberá ser digitada por el operador y no producirá eco en pantalla. En caso de error en la misma el sistema abortará el comando; en otro caso pedirá la nueva palabra de paso, que tampoco tendrá eco en pantalla. Una vez digitada volverá a solicitarla, de tal forma que debemos digitar la misma palabra de paso que anteriormente. Esto se debe a motivos de seguridad del sistema. Si coinciden, la nueva palabra de paso será tomada como la actual para el acceso al sistema. Recuerde esta palabra de acceso, puesto que si se le olvida, deberá solicitar su cambio al administrador del sistema o persona experta encargada de su mantenimiento.

Introducción de comandos

Como dijimos anteriormente, cada vez que se digitan en el teclado los caracteres que componen un comando, el sistema los interpretará y ejecutará una vez que hayamos presionado la tecla RETN (RETURN), produciéndose la interpretación del comando enviado.

Una vez que se ha recibido el prompt del sistema se puede comenzar a introducir comandos, que no son más que requerimientos al sistema para que efectúe un determinado proceso. Como pudimos ver en la primera sesión de trabajo representada, el operador ha digitado el comando **date** y pulsado la tecla RETN, lo que produce una salida a pantalla de la fecha y hora del sistema.

El siguiente comando era **who,** que es una solicitud al sistema para que muestre la lista de usuarios que actualmente están activos en el mismo. La primera columna de la salida corresponde al nombre de usuario que está activo. La segunda es el nombre que el sistema asigna a la conexión utilizada por el usuario (tty es un sinónimo de "teletype" o "terminal"). El resto son los datos de fecha y hora de conexión del usuario. Podríamos también haber usado otra serie de comandos:

```
$ who am i                                Datos del usuario propio
pruebas    tty02  May  17  09:25
$ whom                                    No existe ese comando
whom: not found                           El sistema nos avisa  de
                                          no existencia.
```


Como podemos observar, el sistema nos avisa en caso de que hayamos solicitado la ejecución de un comando que no existe.

Puede ocurrirnos que nuestro terminal actúe de forma extraña, apareciendo dobles los caracteres en pantalla o no posicionando el cursor a principio de línea. Se puede intentar corregir estos defectos apagando y volviendo a encender nuestro terminal. También se puede utilizar el comando **stty** para efectuar el tratamiento inteligente de los caracteres de tabulación. La descripción del funcionamiento de este comando escapa a los propósitos de este libro, pero en caso de necesidad se puede acudir a los Manuales originales de UNIX. También podemos utilizar el comando **tabs** para obtener idénticos resultados.

Control de errores de escritura

En caso de detectar una equivocación al teclear un carácter antes de presionar RETURN, tenemos dos posibilidades de corrección: borrar los caracteres uno a uno o borrar la totalidad del comando digitado. Para este propósito existen dos caracteres que producen un efecto especial para esta corrección:

carácter # → borrado de carácter (erase character)
carácter @ → borrado de la línea digitada (kill character)

Si el operador introduce el carácter de borrado de línea al final de un comando, éste será ignorado completamente y se comenzará a interpretar lo digitado a continuación del carácter "kill".

```
$ ddate@                        Se ignora el comando "ddate"
date                             continuo en nueva linea.
Sat May 17  16:45:50 EDT 1986
$
```

En caso de que el operador digite el borrado de carácter a continuación de cualquier otro carácter, éste será ignorado a la hora de interpretar el comando, con lo cual se pueden hacer correcciones en el comando antes de su envío mediante la tecla RETN:

```
$ dd#atte##e
Sat May 17  16:45:50 EDT 1986
$
```

Los caracteres de borrado de línea y carácter son muy dependientes del sistema y terminal que se estén utilizando. En muchos sistemas el borrado de carácter se ha asimilado a la tecla de

"backspace", que trabaja mucho mejor en los terminales con pantalla de vídeo.

En caso de que deseemos incluir estos caracteres especiales de borrado dentro de un texto, por ejemplo, deberán ir precedidos por el carácter *backslash* "\" (carácter de escape), de tal forma que deberemos digitar "\@" o "\#". Este carácter de escape (\) indica al sistema que el carácter que viene a continuación debe ser tratado de una forma especial.

La introducción de comandos puede ser efectuada a la vez que el proceso en ejecución produce salida a la pantalla. Debido a esto se puede producir la mezcla en la misma de caracteres correspondientes a entrada del operador y salida del proceso. Esta situación no tiene ningún problema, salvo el desorden de salida de caracteres a pantalla. Todo esto es debido a la existencia de un "buffer" de almacenamiento de los caracteres de comandos que el operador está digitando y la posibilidad de multiprocesamiento que implementa el sistema operativo UNIX.

Existe un comando de nombre **stty** que nos permite, por un lado, visualizar los criterios de comunicación con el terminal y, por otro, la modificación de estos criterios.

```
$ stty
speed 9600 baud
interrupt = 'DEL' quit='^\' erase='^H' kill='^U'
even odd -n echo -tabs ff1
$ stty ek
$ stty
speed 9600 baud
interrupt = 'DEL' quit='^\' erase='#' kill='@'
even odd -n echo -tabs ff1
$
```

Al ejecutar el comando **stty** se nos muestra la velocidad de transmisión con el terminal, los caracteres definidos por el sistema para la detención de un proceso, borrado de carácter y línea, salida del sistema, paridad, estado del eco y tabuladores definidos.

La ejecución del comando **stty ek** (ek = erase kill) restaura los caracteres de borrado de carácter y línea al estado normal del sistema. Si volvemos a efectuar **stty** nos muestra el nuevo estado del terminal.

Parada y pausas de un proceso o programa

Mediante la tecla DELETE o RUBOUT se puede producir la parada de un proceso que ha sido lanzado por el usuario a través del envío de un comando determinado. En algunos procesos especiales, como el editor **ed,** esta tecla produce la parada de la tarea que se está efectuando dentro del proceso, pero nos mantiene dentro de él. Mediante el apagado del terminal también se puede producir la parada de numerosos procesos.

Si queremos lograr la parada momentánea de un proceso, podemos teclear **CTRL-s** (recuerde que se debe presionar a la vez la tecla de CTRL y el carácter "s"). Con esto podemos conseguir la parada de la visualización en pantalla de unos datos que por la marcha del proceso van a desaparecer de la misma. Para continuar el proceso normalmente deberemos digitar **CTRL-q.**

Salida del sistema

La forma más normal de acabar una sesión de trabajo con UNIX y salir del usuario en el que estábamos es digitar **CTRL-d,** en lugar de utilizar un comando. Esta secuencia le indica a la Shell que no va a haber más entradas y que se debe dejar el sistema libre para la conexión de otro posible usuario.

Correo entre los usuarios

El sistema operativo UNIX provee una forma de correo-comunicación entre usuarios, soportada por el comando **mail.**

En caso de que el usuario en el que hemos entrado tenga correo pendiente, el sistema lo avisa mediante un mensaje, siendo optativo del operador la visualización o no del mismo. Se obtendrá el correo ejecutando el comando **mail.**

Después de cada mensaje de correo mostrado por el comando, éste espera la acción que se va a desarrollar con el mismo. Hay dos respuestas básicas: una el borrado del correo visualizado, mediante la digitación de la secuencia **d <RETN>**, y otra la visualización del siguiente mensaje pendiente, que se obtiene digitando **RETN.** Otras posibilidades del comando **mail** son:

? p	Vuelve a visualizar el mensaje anterior.
? s <nombre fichero>	Almacena el mensaje en un fichero cuyo nombre le es dado.
? q	Sale del comando **mail** y retorna a la Shell de UNIX.

En caso de que deseemos enviar correo a otros usuarios, deberemos digitar el comando **mail** seguido del nombre de usuario al que se desea enviar. El cursor se posicionará al principio de la siguiente línea y ésta estará lista para la digitación del mensaje.

El final de una línea se marca con RETN y el mensaje continúa en la siguiente línea. Para finalizar el texto del mensaje, digitaremos RETN y CTRL-d, retornando a la Shell.

El comando **mail** también permite el envío de correo a múltiples usuarios, a usuarios de otras máquinas o el envío de cartas ya preparadas.

Comunicación inmediata con otros usuarios

Durante el trabajo normal de una sesión se pueden producir en nuestra pantalla mensajes del tipo: **Message from pedro tty06,** acompañado por un pitido del terminal. Esto nos indica que el usuario "pedro" desea establecer una comunicación con nosotros. Para especificar al sistema que se está de acuerdo, el usuario deberá teclear:

$ write pedro.

Este comando establecerá una doble vía de comunicación entre ambos usuarios, de tal forma que todas las líneas que se digiten en los terminales de los mismos aparecerán en el terminal del otro usuario.

Como el comando **write** no establece ninguna regla de comunicación, los usuarios se deben regir por algún tipo de protocolo. En el caso que vamos a mostrar a continuación, la secuencia **(f)** indica fin de texto de envío (el otro usuario puede enviar mensajes) y la **(ff)** indica que se está listo para el fin de la comunicación:

```
Terminal de pedro                   Terminal de pruebas
$ write pruebas
                                     $ Message from pedro tty06
                                     $ write pedro
Message from pruebas tty02
Recuerdas la cita de hoy? (f)
                                     Recuerdas la cita de hoy? (f)
                                     A las diez®
                                     A las once. Correcto? (f)
A las once. Correcto? (f)
Correcto (ff)
                                     Correcto (ff)
                                     ctrl-d
EOF
ctrl-d
                                     $ EOF
$
```

Mediante la tecla DELETE se puede salir del comando **write.** Note que los errores de digitación corregidos por el usuario "pruebas" no aparecen en el terminal de "pedro", sino que sólo se muestra la línea corregida.

En caso de que se intente la conversación con un usuario que no está conectado al sistema (login) o que estando conectado no desea entablar la conversación, el sistema avisará de ello.

Muchos de los sistemas UNIX poseen un comando especial de nombre **news,** para visualizar las posibles noticias relevantes al sistema, su desarrollo y posibles mejoras e incidencias.

Manual original del sistema UNIX

El manual de usuario **"UNIX Programmer's Manual** describe en detalle el conjunto de funciones y comandos necesarios para el completo manejo del sistema. La Sección 1 del Manual incluye la descripción de comandos UNIX. La Sección 2 trata de los "system calls" del sistema y la Sección 6 incluye los juegos implementados en UNIX. Las demás secciones se ocupan de funciones que pueden ser tratadas por programadores en lenguaje C, formatos de ficheros y mantenimiento del sistema (tarea del administrador del mismo).

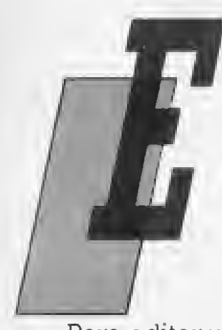
A menudo el Manual original de UNIX está almacenado en el propio sistema y puede ser leído bajo el comando **man** seguido por el nombre del comando cuya sintaxis y descripción se desea:

$ man who	
.....	Visualización de la explicación detallada del comando **who,** sirviendo de ayuda al usuario.
.....	
.....	
.....	
$ man man	Lo mismo del propio comando **man.**
.....	
.....	
$	

El sistema, asimismo, posee un comando que facilita el aprendizaje del sistema operativo, cuyo nombre es **learn.** Este comando incluye el estudio del sistema de ficheros, comandos básicos, editor **ed** del sistema, preparación de cartas y documentos e, incluso, lenguaje C.

CAPITULO III

LOS EDITORES ED Y VI



l editor "ed" es el más sencillo de los editores incorporados por el sistema operativo UNIX. Se trata de un "editor de líneas" en el sentido de que sus comandos han sido concebidos para trabajar sobre una única línea de texto, a diferencia de los "editores de pantalla", preparados para efectuar desplazamientos, búsquedas e inserciones en cualquier punto de la pantalla del ordenador y, por tanto, del texto.

Para editar un fichero denominado "nombre" se ha de teclear:

ed nombre

con lo que el sistema devolvería la respuesta

? nombre

Si se tratase de un fichero de nueva creación, o

nombre
1420

si "nombre" fuese un fichero ya existente, y tuviese un tamaño de 1420 caracteres.

Para grabar un fichero sencillo, como

```
main()
{
        printf("hello, world\n");
}
```

Procederíamos del siguiente modo:

```
$ ed nombre.c
  ? nombre.c
  a
  main()
  {
          printf("hello, world\n");
  }
  .
  w
  q
$
```

El signo "$" corresponde al prompt de la Shell, e indica que está a la espera de recibir un nuevo comando. La línea "ed nombre.c" llama al editor "ed" para modificar el fichero "nombre.c", respondiendo "ed" con la línea "? nombre.c" para indicar que se trata de un nuevo fichero.

La línea que contiene una "a" corresponde al comando "append" (añadir) del editor, indicando que se va a añadir texto al contenido del fichero "nombre.c". A continuación se introduce el texto hasta la aparición de una línea con un punto "." que indica al editor que finaliza la introducción de texto y se pasa del "modo texto" al "modo comando", para seguir introduciendo comandos del editor.

Finalmente, se utilizan los comandos "w" ("write", escribir) para grabar en disco el contenido del fichero, y "q" ("quit", salir) para finalizar la sesión de edición.

Si ahora reeditamos el fichero "nombre.c", el sistema nos responderá con la línea:

```
$ ed nombre.c
  34
```

Indicando que "nombre.c" contiene 34 caracteres. Si tecleamos el comando del editor "1,$p" se mostrará en pantalla el contenido de la primera (1) a la última ($) líneas del fichero.

Para modificar la tercera línea y cambiar "hello" por "hola", procederíamos pulsando "3s/hello/hola/p", con lo que el editor nos respondería con la línea:

```
printf("hola, world \n");
```

El editor "ed" se utiliza en UNIX para mantener y modificar ficheros especiales, que contengan caracteres de control (caracteres ASCII cuyo código es inferior al del carácter espacio en blanco, o ASCII 32 en decimal), y para modificar ficheros en modo no interactivo.

La edición del mismo fichero "nombre.c" mediante el editor vi se realizaría como:

```
$ vi nombre.c
  "nombre.c" [new file]
```

Borrándose la pantalla, y presentando el carácter " " de la primera a la última línea, en la que aparecería el nombre del fichero y el texto "[new file]", indicando que se trata de un nuevo fichero.

Seguidamente se introduciría el comando "a" ("append" añadir) y comenzaría la introducción del texto. Para finalizar la misma, se pulsa la tecla ESCAPE "ESC" y se pasa al "modo comando", tecleando posteriormente "wq" para grabar el fichero y finalizar la edición.

Para el movimiento de una línea a otra se pueden utilizar las flechas dispuestas sobre el teclado, en lugar de referirse a una línea en particular como en el caso del editor "ed".

"vi" es tan sólo un programa editor y no resulta, por tanto, tan completo como pueda ser un tratamiento de textos como el "Wordstar" de CP/M. Permite el manejo de instrucciones de la Shell o el empleo de comandos de "ed" para aumentar su flexibilidad y repertorio de comandos.

Por razones de espacio, no profundizaremos en el manejo de los editores "ed" y "vi", remitiendo al lector interesado al apéndice A, donde se incluye un amplio resumen de comandos de UNIX, figurando allí las opciones más frecuentes de estos editores.

CAPITULO IV

FICHEROS DEL SISTEMA UNIX

Todo en el Sistema UNIX son ficheros. Es la idea más sencilla y a la vez más clarificante que se pueda tener en un principio sobre dicho sistema operativo. Las mayores discusiones en los inicios de la creación del sistema UNIX se centraron en la estructura interna que mantendría el sistema organizativo de ficheros, de tal forma que fuese a la vez potente y sencilla su utilización. Se eligió como base de esta filosofía el implementar un sistema basado en un **pequeño conjunto de muy buenas ideas,** que diese como resultado un poderoso sistema de ficheros.

En este capítulo se cubrirán algunos detalles sobre la estructura, creación y organización de ficheros del sistema, directorios, permisos de acceso. Se incluirá la descripción de numerosos comandos del sistema que efectuarán tareas de mantenimiento y manipulación de ficheros o que, en definitiva, mantienen una fuerte ligazón con el sistema de ficheros UNIX.

Nociones básicas sobre ficheros

La definición más clara y sencilla de un fichero es la de una secuencia de bits, entendiendo por bít la unidad más pequeña de información procesada por cualquier equipo que trabaje bajo el sistema operativo UNIX. En principio, la estructura de la información que esté almacenada en el fichero sólo es relevante para el programa que procesará ese fichero. Para el sistema operativo UNIX, una serie de caracteres tecleados en la consola, cualquiera

de las impresoras del sistema, los conjuntos de información (ficheros de datos) almacenados en el disco duro, una cinta magnética (streamer) o los datos que fluyen a través de los **pipes** del sistema son considerados como ficheros, teniendo un tratamiento acorde con la estructura y utilización de cada uno.

La mejor forma de conocer y profundizar en el sistema de ficheros UNIX es crear uno y efectuar sobre él una serie de procesos. Hagámoslo mediante el comando del sistema **ed** (entrada al editor "ed").

```
$ ed
a
ABCDEFGHIJKLMNO
0123456789
.
w texto
27
q
$ ls -l texto
-rw-r--r-- 1 prueba     27 May 12 09:54 texto
$
```

El fichero "texto" contiene 27 bytes, que son los caracteres tecleados por el usuario que lo ha creado. Para visualizar el contenido del fichero utilizaremos el comando **cat:**

```
$ cat texto
ABCDEFGHIJKLMNO
0123456789
$
```

Vamos a visualizar el contenido interno (en bytes) del fichero "texto". Mediante el comando **od** (octal dump) obtenemos una representación visible de todos los bytes que componen el fichero:

```
$ od -c texto
0000000 A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  \n
0000020 0  1  2  3  4  5  6  7  8  9  \n
0000030
      $
```

Mediante la opción **—c** se interpretan los bytes como si fuesen "caracteres". La opción **—x** nos mostrará los bytes en números hexadecimales (base 16):

```
$ od -cx texto
0000000 A  B  D  E  F  G  H  I  J  K  L  M  N  O  \n
        41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 0A
0000010 0  1  2  3  4  5  6  7  8  9  \n
        30 31 32 33 34 35 36 37 38 39 0A
0000020
```


Los dígitos numéricos que aparecen en la parte izquierda de la salida son las posiciones que ocupan los bytes dentro del fichero, *en notación hexadecimal*. Fijémonos que cada línea de salida representa el contenido de 16 bytes del fichero.

Al final de cada una de las líneas del fichero "texto", aparece un carácter " **\n**" que representa el "final de línea" **(newline)** que el usuario ha definido mediante la tecla <RETURN>. Por convención basada en el desarrollo del Lenguaje C, este carácter se representa mediante la notación " **\n**". Internamente se almacena el carácter "12" en octal o el "0A" en hexadecimal. El ejemplo de este carácter especial es uno de los muchos existentes:

- escape (escape)
- backspace (espacio atrás) **\b** Octal 10 Hex. 08
- tab (tabulador) **\t** Octal 11 Hex. 09
- carriage return (retorno de carro) **\r** Octal 15 Hex. 0D

Es muy importante que sepamos distinguir entre el contenido interno de un fichero, o sea, los bytes almacenados en el mismo, y la interpretación que de ellos se hace dependiendo de las situaciones o los procesos que traten dicho contenido.

Supongamos que en un momento de nuestro trabajo con el sistema tecleamos el carácter **backspace** y que dicho carácter está definido como **borrado de carácter;** el kernel interpreta que se pretende borrar el carácter anteriormente tecleado, con lo cual nos desaparecerán de la pantalla ambos caracteres. Si, en cambio, el usuario teclea la secuencia **<backspace>,** el kernel interpreta que se desea almacenar dentro de nuestro fichero de entrada (en este caso la consola) el literal **backspace** y el byte representado por 08 en hexadecimal será almacenado, haciendo eco del mismo en la consola (salida).

Si utilizamos el comando **print** sobre un fichero que contenga un carácter especial **backspace,** con salida a consola, producirá el movimiento del cursor un espacio a la izquierda. Si, en cambio, sobre el mismo fichero se utiliza el comando **od** con la misma salida, nos aparecerá como un byte de representación 08 en hexadecimal o 10 en octal.

El sistema operativo UNIX no posee **registros,** ni contadores de bytes en registros, ni ningún otro byte que no sean los almacenados por nuestro propio programa de mantenimiento del fichero. Esta es una de las reglas básicas que se establecieron durante el desarrollo del sistema de ficheros que soportaría el UNIX.

Por otro lado, vemos que al final del fichero no aparece ningún carácter especial indicativo de **fin de fichero,** sino que simplemente se detiene la representación del mismo. El sistema indica que se ha terminado el fichero simplemente porque no hay más datos en el mismo. El kernel mantiene tablas indicativas de

la longitud total del fichero, de tal forma que los procesos encontrarán el fin del fichero cuando hayan procesado la totalidad de bytes del mismo.

Localización de ficheros

La mayoría de los comandos que vamos a explicar en las siguientes secciones del capítulo, así como en muchos otros, incluyen la posibilidad de especificar unos nombres de ficheros que se le pasan como argumentos al comando. Estos nombres de ficheros pueden ir **explícitos**, o sea, con su nombre completo, o definidos con una serie de caracteres especiales, que van a obligar a efectuar una búsqueda de ficheros, basándose en unas reglas que están especificadas en el núcleo del sistema. Veamos una tabla de los caracteres especiales sustitutivos dentro del nombre de un fichero, junto con serie de ejemplos clarificadores:

TABLA DE CARACTERES DE MASCARA UNIX (WILD CARDS)	
?	En cualquier posición indica que es válido cualquier carácter, pero sólo uno.
*	Indica cualquier cadena de caracteres a partir de esa posición, incluso la cadena nula.
[abc]	En la posición dada podrán existir los caracteres "a", "b" o "c" y, en general, cualquiera de los caracteres situados dentro de los corchetes.
[^abc]	Igual que la anterior, pero son válidos todos los caracteres excepto los contenidos dentro de los corchetes.
[a-b]	En esa posición es válido cualquier carácter que esté alfabéticamente entre el carácter "a" y el carácter "b".
^	Fuerza a que los caracteres de la máscara estén al principio del nombre de fichero.
$	Fuerza a que los caracteres de la máscara estén al final del nombre del fichero.

Veamos unos ejemplos de utilización de estas máscaras:

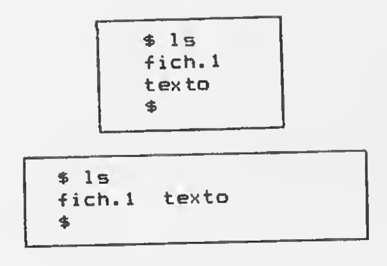
nom.?	⟶	válidos "nom.<carácter>" Ejemplo: "nom.a" "nom.1" "nom.p" "nom.x". No son válidos "nombre" "nom.12" "no.1" "nom.ab"
pro.*	⟶	válidos "pro.<cadena>" Ejemplo "pro.c" "pro.program" "pro.12". No son válidos "nom.1" "program" "pro12" "pr.c"
fic.[123]	⟶	válidos únicamente "fic.1" "fic.2" "fic.3"
fic.[^123]	⟶	válidos todos los "fic.<carácter>", excepto los "fic.1" "fic.2" y "fic.3"
uno[a-d]	⟶	válidos "unoa" "unob" "unoc" y "unod"

Vamos a crear dos ficheros (siempre dentro del usuario de trabajo "prueba") mediante el comando **ed:**

```
$ ed
a
Texto del ducumento uno
.
w texto
24
q
$ ed
a
El sistema UNIX en estudio
.
w fich.1
27
q
$
```

Recordemos que el contador de caracteres mostrado por el comando incluye el carácter de fin de línea **(newline)**, que es la forma que tiene el sistema de representar la digitación de la tecla RETURN.

El comando **ls** nos muestra la lista de los nombres de los ficheros que mantienen el usuario o el sistema, no el contenido de los mismos:



Coincide con los dos ficheros creados por el usuario. Son válidos los dos sistemas de representación que aparecen; dependiendo de la versión de sistema UNIX con la que se trabaje usaremos una u otra. Los nombres están automáticamente ordenados alfabéticamente. Como muchos de los comandos UNIX, **ls** posee opciones que pueden ser utilizadas para cambiar el modo de acción del mismo. Las opciones deben ir a continuación del nombre del comando y, normalmente, van precedidas del carácter "—".

La opción **—t** produce una ordenación de los nombres de ficheros por la hora de la última modificación de los mismos. Los de cambio más reciente son colocados al principio.

La opción **—l** produce una salida con información mucho más completa sobre los ficheros.

```
$ ls -l
total 2
-rw-r--r-- 1 prueba     27  May 17  18:21  fich.1
-rw-r--r-- 1 prueba     24  May 17  18:20  texto
```

El primer mensaje de salida ("total 2") nos indica el número total de bloques de disco utilizados para almacenar la información de los ficheros que han sido procesados por el comando. Un bloque está constituido por 512 ó 1024 caracteres. La cadena "-rw-r--r--" nos da información de los permisos de acceso a los ficheros. En este caso el propietario del fichero ("prueba") tiene permiso de lectura (r) y escritura (w), pero el resto de usuarios solamente tienen permiso de lectura. La siguiente información que observamos en la salida ("1") es el número de enlaces **(links)** del fichero. La cadena "prueba" nos indica el nombre de usuario al que pertenece el fichero, es decir, el usuario que lo creó. La siguiente cadena de salida ("27" y "24") corresponde al número de caracteres que componen cada fichero (bytes) y que vemos que coincide con el obtenido por el comando **ed** en la creación de ambos ficheros. A continuación muestra la fecha y hora de la última modificación del fichero y, por último, el nombre del mismo.

Las opciones del comando **ls** (así como las de muchos otros) pueden ser agrupadas, de tal forma que el comando **ls —lt** produce una salida con información completa de los ficheros y éstos ordenados por fecha-hora de última modificación.

La opción **—u** nos da información del momento del último uso del fichero (no cambio), **—r** produce la inversión del orden de salida de los ficheros procesados por el comando.

Después de las opciones del comando **ls** podemos incluir uno o varios nombres de fichero, de tal forma que la información visualizada por el comando pertenecerá a la lista de ficheros dada:


```
$ ls -l texto
-rw-r--r-- 1  prueba        24  May 17 18:20 texto
$ ls -l fich.1
-rw-r--r-- 1  prueba        27  May 17 18:21 fich.1
$ ls -l tex
tex not found
$
```

Como podemos observar, en caso de que no exista el fichero solicitado el comando nos muestra un mensaje de aviso. Veamos a continuación una tabla de las opciones más importantes del comando **ls.** Tenga en cuenta que en esta parte del capítulo aparece la noción de directorio, que será más tarde explicada; quede aquí su reseña a nivel de su utilización en el comando **ls:**

ls [—ltasdru] <nombres de fichero>

- —l lista en formato completo (largo) de información;
- —t lista por orden de fecha-hora de última modificación. Primero los más recientes;
- —a lista todos los ficheros incluyendo directorios;
- —s visualiza el tamaño de los ficheros en bloques. 1 bloque = 512 caracteres (bytes);
- —d informa sobre la situación de directorios;
- —r lista en orden inverso al normal;
- —u lista por orden de fecha-hora de última utilización de los ficheros. Primero el más reciente.

Unos ejemplos serían:

```
$ ls -a
.  ..  .profile  fich.1  texto
$ ls -s
total 2  1 fich.1  1 texto
$ ls -r
texto  fich.1
$ ls -u
fich.1  texto
```

Vemos en la opción **—a** que nos aparecen tres ficheros "extraños". El representado por **"."**, nos indica el directorio actual en el que nos encontramos. El **".."** corresponde al directorio padre del actual, es decir, el directorio del que proviene o cuelga. Y el fichero **".profile"** es uno de los ficheros "invisibles" del usuario; contiene una serie de parámetros importantes para el control del usuario.

El comando **ls** no suele responder con mensajes de error, in-

cluso en el caso de una digitación incorrecta de las opciones. Cuando no se le pasan argumentos de nombres de ficheros lista todos los ficheros del directorio actual. En caso de digitar la opción —l del comando ésta imprime, entre otras informaciones, una lista de 10 caracteres, de los cuales el primero indica el tipo de fichero que se ha listado y los restantes 9 los permisos de acceso al mismo, que serán explicados posteriormente. Los tipos posibles de ficheros, indicados por el primer carácter de la cadena, son los siguientes:

- d la entrada es un directorio;
- b la entrada es un fichero especial de tipo bloque;
- c la entrada es un fichero especial de tipo carácter;
- — la entrada es un fichero del tipo ordinario. Fichero normal de usuario.

Listado del contenido de un fichero

Existen numerosas formas para observar o visualizar el contenido de un fichero. Una posibilidad es usar el editor del sistema **ed:**

```
$ ed texto
24                          Indica el número de caracteres
1,$p                        Listar líneas desde la 1 hasta
Texto del documento uno     la última
q                           Salida del editor
$
```

Llamamos al comando **ed** pasándole el nombre del fichero que queremos editar ("texto"); nos indica el número de caracteres (bytes) que lo componen.

Existen numerosos casos en los que no es viable el uso de un editor para la visualización de la información almacenada en un fichero, por ejemplo, en el caso de que el tamaño del mismo sea superior al máximo procesado por el editor o que se desee la visualización de más de un fichero. Para estos casos existen varias alternativas que veremos a continuación.

El comando **cat** es el más simple de todos los posibles para visualizar información de ficheros. Este comando lista el contenido de todos los ficheros que se le hayan pasado como argumentos:

```
$ cat texto
Texto del documento uno
$ cat fich.1
```



```
El sistema UNIX en estudio
$ cat texto fich.1
Texto del documento uno
El sistema UNIX en estudio
$
```

El nombre del comando procede de la palabra con**cat**enación. Se produce la concatenación de los ficheros pasados como argumentos, sobre la salida (por defecto pantalla). La salida del comando se producirá sin ningún tipo de pausa, por lo que si el contenido del fichero es grande, se deberá pulsar **ctrl-s** para parar la salida y mantener la visualización que nos interesa.

El comando **cat** sólo sirve para la visualización del contenido de ficheros que estén compuestos por caracteres ASCII o bien caracteres que sean imprimibles. Si, por ejemplo, se lista un fichero con caracteres de control, el efecto que pueden causar en nuestro terminal puede ser bastante extraño. El comando **cat** puede visualizar un mensaje de error, indicando que no encuentra el fichero solicitado:

```
$ cat salvador
cat: can't open salvador
$
     o bien dependiendo de la versión:
$ cat salvador
salvador: cannot open
$
```

En todos los sistemas UNIX existe un comando que permite acomodar la salida a las características del terminal, de tal forma que nos permita la visualización del contenido por páginas. El nombre del comando depende del sistema y puede ser **page, pg** o **more.** Queda al usuario el investigar la existencia en su sistema de estos comandos.

Existe otro comando, de nombre pr, que lista el contenido de uno o varios ficheros, acomodándose a las características de las impresoras del sistema. Es decir, lista páginas de 66 líneas (11 pulgadas) con la fecha y hora de última modificación del fichero, numera las páginas y coloca el nombre del fichero en el principio de cada una. Entre cada uno de los ficheros a listar efectúa un salto de página, de tal forma que cada fichero comienza al principio de una página.

```
$ pr texto fich.1

May 17  18:20  1986  texto    Page 1

Texto del documento uno
          (62 líneas en blanco)
```

```
May 17  18:21  1986  fich.1   Page 1

El sistema UNIX en estudio
          (62 líneas en blanco)
$
```

Mediante este comando se puede producir una salida en multicolumnas o en paralelo. Por ejemplo, **pr —3 fich.1 fich.2 fichero** efectúa una salida en formato de tres columnas. La opción **—m** asigna la salida en columas paralelas. Para más información sobre el comando se debe acudir al Manual original de UNIX (UNIX Programmers Manual).

Debemos hacer notar que este comando no efectúa ningún tipo de arreglo en las líneas del fichero, ni justifica márgenes. Estas tareas son ejecutadas por comandos más complejos y de mayor potencia como son **nroff** y **troff,** cuya discusión escapa a las intenciones de este libro.

El comando **pr** no produce ningún tipo de mensaje de error en caso de no encontrar los ficheros a imprimir ó que se digiten opciones incorrectas. Existe otro detalle acerca del comando. En cuanto se comienza a imprimir el listado en el terminal, se suprimen todas las comunicaciones establecidas mediante el comando **write,** de tal forma que se evita la alteración del formateo del listado. Veamos las opciones más comunes del comando:

- -<n> — la salida se imprime en formato de "n" columnas;
- +<n> — imprime el fichero a partir de la línea "n" especificada, inclusive;
- -h <cadena> — indica cambio de la cabecera. La <cadena> es ahora la cabecera de cada página;
- -w<n> — la anchura de cada página del listado será de "n" caracteres, en lugar de 72, tomado por defecto;
- l<n> — cambia el número de líneas por página al valor "n", en vez de 66 líneas tomadas por defecto;
- t — el listado evita (salta) las 5 líneas de cabecera y de fin de página;
- -s<carácter> — en vez de separar las columnas por el carácter de tabulación, éstas son separadas por el carácter especificado en <carácter>;
- -m — imprime todos los ficheros especificados en columnas paralelas y simultáneamente.

Renombrado, copia y borrado de ficheros

Una de las tareas más normales en el trabajo con un sistema operativo es el cambio del nombre de un determinado fichero o renombrado del mismo. El comando que lo efectúa se denomina **mv** y su sintaxis es:

mv <nombre antiguo> <nombre nuevo>

```
$ mv texto archivo
$ ls
archivo
fich.1
$ cat texto
cat: can't open texto
$
```

Hemos cambiado el nombre del fichero "texto" a "archivo". El contenido del fichero permanece inalterado. Posteriormente ejecutamos el comando **ls,** mostrándonos una lista de los ficheros del usuario en la que no aparece el fichero "texto" y sí lo hace "archivo". Si intentamos listar el contenido del fichero "texto" mediante el comando **cat,** éste nos avisará con un mensaje de error indicativo de su no existencia. Tenga muy en cuenta que si pretendemos renombrar un fichero con un nombre de otro que ya existe, éste será borrado del sistema y pasará a contener la información del fichero que se ha renombrado.

En el apartado dedicado a directorios UNIX volveremos sobre este comando para ver la posibilidad de "mover" ficheros dentro de los caminos (pathnames) del árbol de directorios del sistema y del usuario.

Para efectuar copias de ficheros, es decir, proceder a una duplicación de los mismos con nombres distintos, existe el comando **cp,** que no hace sino mantener varias versiones de un determinado fichero:

```
$ cp archivo salva
$ cat salva
Texto del documento uno
$ cp fich.1 fich.2
$ cat fich.2
El sistema UNIX en estudio
$
```

Se han generado dos copias, una del fichero "archivo" sobre otro de nombre "salva" y otra del fichero "fich.1" sobre "fich.2", y que poseerán exactamente el mismo contenido, como hemos visto mediante el comando **cat salva** y **cat fich.2.**

```
$ cp archi.1 files
cp: can't open archi.1
$ cp archivo archivo
cp: cannot copy file to itself
$ cp -s archivo nuevo
Usage: cp f1 f2 or cp f1..fn d2
```

Podemos ver que el comando **cp** avisa en caso de no encontrar el fichero que va a ser copiado. Igualmente, en caso de intentar copiar un fichero sobre sí mismo, nos avisa de tal situación. Por último, el comando no posee ningún tipo de opciones y, en caso de digitar alguna, nos avisa de que no es correcta la sintaxis del comando.

Existe otro comando que permite el borrado de ficheros del usuario o sistema, de nombre **rm** y cuya sintaxis es:

rm <lista de ficheros>

```
$ rm fich.2
$ rm texto
rm: texto nonexistent
$ rm salva
$
```

Hay que tener mucho cuidado con los ficheros que se van a borrar, puesto que el comando **rm** no solicita la conformidad para tal borrado. Existe una opción del comando, de nombre **-i**, que solicita interactivamente la conformidad para cada uno de los ficheros de la lista de argumentos del comando. Esta opción se debe usar cuando no se sepa exactamente la lista de ficheros a borrar, por utilizar caracteres de máscara. El borrado de ficheros se efectúa en "modo silencioso", es decir, no se visualiza ningún tipo de mensaje que avisa de que el borrado ha sido ejecutado. Veamos las opciones del comando **rm:**

rm [-rif] <ficheros>

- -f serán borrados también aquellos ficheros protegidos contra escritura.
- -r borra recursivamente todos los ficheros de un directorio e incluso el propio directorio.
- -i solicita confirmación para el borrado de cada fichero digitado.

Mensajes de error del comando:

```
$ rm arch.1
rm: arch.1 nonexistent
```



```
$ rm -v fic.1 fic.2
rm: unknown option -v
$
```

Debemos tener mucho cuidado con la opción **-r**, puesto que nos borrará, sin pedir confirmación, todos los ficheros del directorio actual y el propio directorio. En este caso es muy conveniente usar la opción **-i** para la solicitud de la confirmación de cada borrado de fichero.

Normas adicionales sobre nombres de ficheros

Vamos a marcar unas normas para que los nombres dados a los ficheros sean de alguna forma legales al sistema y al usuario. En primer lugar, la longitud máxima del nombre de un fichero es de catorce (14) caracteres. Si sobrepasamos esta longitud el sistema trunca el nombre a los catorce primeros caracteres. En segundo lugar, los caracteres que componen el nombre del fichero pueden abarcar una gran parte del juego de caracteres del sistema, pero el sentido común nos hace pensar en eliminar una serie de ellos que pueden dar lugar a confusiones para el usuario y el sistema. Por ejemplo, si damos a un fichero el nombre "-t", al intentar localizarlo mediante el comando **ls -t**, el sistema interpretará que se desa una lista de todos los ficheros del usuario clasificada en orden de tiempo de última modificación y no que es el fichero concreto deseado por el usuario. Podemos deducir que nunca debemos asignar como primer carácter del nombre de un fichero el "-". Para intentar evitar todos estos problemas les aconsejamos que solamente sean usados caracteres alfabéticos, numéricos y los símbolos "." (punto) y "_" (subrayado) en la composición de los nombres de ficheros, hasta que el usuario esté completamente seguro de la situación que se puede crear en la asignación de nombres a los mismos.

Comandos de procesamiento de ficheros

Una vez que hemos visto la forma de crear ficheros, localizarlos y listar su contenido, vamos a desarrollar una serie de comandos de uso general y que ofrecen una potente herramienta de trabajo sobre ficheros. Para la discusión de los mismos vamos a crear un fichero con un determinado contenido:

```
$ ed
a
Con cien cañones por banda
```

```
  viento en popa a toda vela
No cruza el mar sino vuela
  un velero bergantín.
Poema viento.
.
w poema
119
q
$
```

El primer comando que vamos a desarrollar efectúa la cuenta del número de líneas, palabras y caracteres que componen uno o más ficheros. Su nombre es **wc** (word counter):

```
$ wc poema
        5        22        119        poema
$
```

Una palabra se puede definir como el conjunto de caracteres que no contiene ningún blanco, tabulador o "newline". El fichero "poema" contiene 5 líneas, 22 palabras y 119 caracteres. En caso de que al comando **wc** se le pase como argumentos una lista de ficheros (más de uno), nos listará la cuenta para cada uno de los ficheros de la lista, así como el número total de caracteres, palabras y líneas en la totalidad de los mismos:

```
$ wc poema archivo
              5        22        119     poema
              1         4         24     archivo
              6        26        143     total
$
```

Veamos las opciones del comando:

wc [-lwc] <ficheros>

- l cuenta únicamente las líneas del fichero
- w cuenta únicamente las palabras del fichero
- c cuenta únicamente los caracteres del fichero

Mensajes de error del comando:

```
$ wc -n archivo
archivo
$ wc -lv archivo
    1  archivo
$ wc +c archivo
wc: can't open +c
    1      4      24    archivo
$
```


En el primer caso, al ser inválida la opción y existir el fichero, no muestra ningún tipo de información en la cuenta del mismo; en el segundo caso ignora la opción inválida y muestra la información correspondiente a la opción válida; en el tercer caso nos avisa de la no existencia del fichero.

Otro importante comando se denomina **grep.** Se encarga de buscar dentro de los ficheros que se le pasan como argumentos, las líneas que contengan una cadena determinada que se le da al comando como parámetro. Es decir, pasada una determinada cadena de caracteres nos visualiza las líneas del fichero(s) que contengan dicha cadena:

```
$ grep viento poema
  viento en popa a toda vela
Poema viento
$ grep documento archivo
Texto documento uno
$
```

Sobre el fichero "poema" hemos localizado todas las líneas que contienen la cadena "viento", y sobre el fichero "archivo", las que contienen "documento". El comando **grep** también nos localiza las líneas que "no" contienen la cadena especificada. Esto se consigue mediante la opción **-v:**

```
$ grep -v viento poema
Con cien cañones por banda
No cruza el mar sino vuela
  un velero bergantín.
$
```

El comando **grep** posee numerosas opciones que efectúan tareas diferentes basadas en la norma fundamental del comando, que es la búsqueda de líneas de ficheros que contengan una determinada cadena de caracteres. Se puede efectuar la búsqueda en varios ficheros, contar líneas y caracteres y numerar líneas. Veamos las opciones más comunes e importantes del comando:

grep <opciones> <cadena> <ficheros>

- -v Muestra todas las líneas excepto las que contienen la <cadena> dada.
- -c Muestra todas las líneas que contienen la <cadena> digitada.
- -l Muestra los nombres de los ficheros que contienen la < cadena> digitada.

- -n Aparte de mostrar las líneas visualiza el número de línea.
- -h No visualiza el nombre de los ficheros.
- y Da igual tratamiento a las mayúsculas y a las minúsculas.
- -e Permite comentar la <cadena> por el carácter "-".

Otro comando de gran importancia es **sort,** que ejecuta una clasificación de las líneas del fichero de entrada por orden semi-alfabético. Es decir, para clasificar dos líneas, compara el primer carácter de cada una, si son iguales al segundo, y así hasta que exista una diferencia que permita establecer la clasificación entre ambas:

```
$ sort poema
  un velero bergantín.
  viento en popa a toda vela
Con cien cañones por banda
No cruza el mar sino vuela
Poema viento
$
```

La clasificación es línea a línea. El orden natural de clasificación incluye antes los blancos, después las letras mayúsculas y, por último, las minúsculas, por lo que no es estrictamente una clasificación alfabética. El comando **sort** posee múltiples opciones para controlar el orden de la clasificación: posibilidad de orden inverso, numérico, ignorando blancos repetidos, clasificando campos dentro de la línea, etc. Veamos a continuación las opciones más comunes e importantes del comando:

sort <opciones> <ficheros>

- -r Invierte el orden normal de clasificación.
- -n Clasifica en orden numérico.
- -nr Clasifica en orden numérico inverso.
- -f Considera igual las mayúsculas y minúsculas.
- +n Comienza a clasificar en el campo n+1.

Otro interesante comando para visualizar contenidos de un fichero es **tail.** Este comando visualiza las últimas diez (10) líneas de un fichero. En ficheros como "poema" carece de interés, pero en ficheros mayores puede ser muy importante. El comando **tail**

posee opciones para especificar el número de líneas a visualizar, de tal forma que podemos seleccionar las mismas:

```
$ tail archivo
Texto documento uno
$ tail -1 poema
Poema viento
$ tail +3 poema
No cruza el mar sino vuela
  un velero bergantín.
Poema viento
$
```

Mediante la opción **-**<n>, donde <n> debe ser un dato numérico, nos visualiza las "n" últimas líneas del fichero. En este caso hemos ejecutado **tail -1 poema,** que nos mostrará la última línea del fichero "poema". La opción **+<n>**, donde <n> tiene el mismo significado anterior, visualiza a partir de la "n" línea del fichero. El comando ejecutado **tail +3 poema** muestra todas las líneas de fichero "poema" a partir de la tercera inclusive. Veamos las opciones del comando:

tail [+-]<n><opciones> <fichero>

+ Indica comienzo de listado a partir de la <n> línea del fichero, inclusive.
- Indica el comienzo de listado a partir de <n> líneas del fin del fichero.

OPCIONES:

- l El número <n> se refiere a líneas. Esta opción es por defecto.
- c El número <n> se refiere a caracteres.
- b El número <n> se refiere a bloques de ficheros (grupos de 512 caracteres).

Los últimos dos comandos que vamos a desarrollar permiten verificar comparaciones y diferencias entre ficheros. Para ello vamos a crear otro fichero:

```
$ ed
a
Con cien canones por banda
  viento en popa a toda vela
No cruza el mal sino vuela
  un velero bergantín.
```

```
Poema viento
.
w poema.dos
119
q
$
```

Vemos que entre ambos ficheros no existe mucha diferencia, sólo dos palabras (canones y mal) son distintas. El comando **cmp** encuentra la primera línea en la que difieren dos ficheros y nos indica la posición primera de diferencia entre ambos:

```
$ cmp poema poema.dos
poema poema.dos differ: char 12, line 1
$
```

El comando **cmp** nos indica que los dos ficheros son idénticos hasta el carácter 12 de la primera línea. No nos dice las diferencias entre ambos, sino solamente que está en la primera línea. Este comando es muy interesante para casos en los que deseemos verificar la igualdad total entre dos ficheros.

Existe otro comando de nombre **diff** que nos da información de las líneas que son distintas, las que faltan y las que sobran, en la comparación de ambos archivos:

```
$ diff poema poema.dos
1c1
<     Con cien cañones por banda
---
>     Con cien canones por banda
3c3
<     No cruza el mar sino vuela
---
>     No cruza el mal sino vuela
$
```

El comando nos indica que la línea 1 del primer fichero ("poema") está cambiada respecto de la línea 1 del segundo ("poema.dos") e igualmente con la tercera.

El comando **cmp** funciona con multitud de tipos de ficheros, mientras que el **diff** solamente funciona con ficheros del tipo texto. El comando **diff** se usará cuando se deseen conocer las diferencias exactas entre dos ficheros. Evidentemente, es más rápido el comando **cmp**, pues solamente busca la primera diferencia. Veamos las opciones del comando **diff:**

[illegible] [illegible]he] <fichero1> <fichero2>

- [illegible] la salida de las diferencias entre los dos ficheros se efectúa en orden inverso.

- -h localiza pequeñas diferencias, pero actúa de forma mucho más rápida, y es muy conveniente en grandes ficheros. No es compatible con **ed.**
- -b Ignora los tabuladores y blancos de final de líneas, y trata como un solo blanco los repetitivos.
- -e Admite las órdenes de añadir, borrar y modificar líneas, con los mismo comandos que el editor **ed.**

Mensajes de error:

```
$ diff archivo arc.1
diff: cannot open arc.1
$ diff -x archivo texto
$
```

El comando **diff** avisa en caso de que alguno de los dos ficheros no exista. En caso de digitar una opción inválida, el comando no nuestra ningún mensaje de error.

TABLA DE COMANDOS más comunes en el manejo de Ficheros	
ls	Lista nombres de todos los ficheros del directorio actual del usuario.
ls <ficheros>	Lista sólo los nombres de la lista.
ls -t	Clasificados en orden de fecha-hora de última modificación de los ficheros.
ls -l	Información completa de todos los ficheros. También ls -lt.
ls -u	Clasificada por fecha-hora de último uso de los ficheros. También ls -lu y ls -lut.
ls -r	Invierte el orden de salida. También ls -rt, ls -rlt, etc.
ed <fichero>	Edición del fichero indicado.
cp <fic.1> <fic.2>	Copia <fic.1> sobre <fic.2>. Copia encima del <fic.2> si éste ya existe.

TABLA DE COMANDOS más comunes en el manejo de Ficheros	
mv <fic.1> <fic.2>	Mueve (renombra) <fic.1> a <fic.2>. Copia sobre <fic.2> si éste ya existe.
rm <ficheros>	Borra ficheros indicados irrevocablemente.
cat <ficheros>	Lista (visualiza) el contenido de los ficheros indicados.
pr <ficheros>	Lista el contenido de los ficheros con páginas de 66 líneas y cabecera.
pr -n <ficheros>	Igual, pero en formato de "n" columnas.
pr -m <ficheros>	Igual pero en múltiples columnas. Un fichero al lado del otro.
wc <ficheros>	Cuenta líneas, palabras y caracteres en cada fichero y en su total.
wc -l <ficheros>	Cuenta solamente líneas.
grep <cad> <ficheros>	Lista líneas de los ficheros que contengan la cadena <cad>.
grep -v <cad> <ficheros>	Lista líneas de los ficheros que no contengan la cadena <cad>.
sort <ficheros>	Clasifica los ficheros por orden alfabético de líneas.
tail <fichero>	Lista las últimas 10 líneas del fichero.
tail -n <fichero>	Lista las "n" últimas líneas del fichero.
tail +n <fichero>	Lista las líneas del fichero comenzando en la línea "n".
cmp <fic.1><fic.2>	Indica la primera diferencia localizada entre los dos ficheros.
diff <fic.1><fic.2>	Lista todas las diferencias entre los dos ficheros.

Directorios

El sistema operativo UNIX distingue a cada uno de los ficheros del sistema, además de por su nombre, por la agrupación de los mismos dentro de entidades denominadas directorios. Es similar a la forma en que los libros son colocados en una biblioteca dentro de librerías. La biblioteca es el sistema, las librerías son los directorios y los libros son los ficheros.

Cada usuario del sistema (definido entre otras cosas por su nombre) tiene un directorio particular, denominado directorio del usuario o directorio corriente. El usuario puede estar trabajando en otro directorio, pero siempre poseerá el suyo propio. A menos que se efectúe un cambio de directorio, cualquier fichero creado por el usuario pertenecerá al directorio del mismo. Al efectuar la conexión al sistema (login) se produce la entrada en el directorio del usuario, que posteriormente veremos cómo se representa.

Un directorio, además de poseer ficheros normales, puede contener otros directorios. La forma normal de representación de esta situación es la de un árbol de ficheros y directorios. El sistema da la posibilidad de moverse dentro de este árbol y localizar cualquier fichero del sistema iniciando la búsqueda en la raíz **(root)** del árbol. El comando básico de localización del directorio actual en el que el usuario está trabajando se denomina **pwd.**

```
$ pwd
/usr/pruebas
$
```

El comando visualiza el directorio actual de trabajo, que coincide con el directorio del usuario "pruebas". Como podemos ver el directorio del usuario es un directorio dentro del directorio **usr,** que a su vez es un directorio del directorio base del árbol, denominado **(root)** (raíz) y que se representa por "/". Este carácter se usa como separador de los directorios y ficheros del árbol. También existe la limitación de 14 caracteres para el nombre de un directorio. En casi todos los sistemas UNIX, los usuarios del mismo cuelgan del árbol en la forma **/usr.** Veamos unos ejemplos del comando **ls** incorporando la noción de directorios:

```
$ ls /usr/pruebas
archivo
fic.1
$ ls /usr
antonio
director
fuentes
maestro
pruebas
```

```
$ ls /
bin
boot
dev
etc
lib
tmp
unix
usr
$
```

El comando **ls /usr/pruebas** lista los ficheros que existen en el usuario "pruebas", es decir, dentro de su directorio normal. El comando **ls /usr** lista los nombres de los directorios de usuarios definidos en el sistema o, en general, la lista de ficheros y directorios que cuelgan del árbol desde la base **/usr.** Por último, el comando **ls /** lista la totalidad de ficheros y directorios que cuelgan desde la raíz **(root)** del árbol. Veamos unos ejemplos con el comando **cat:**

```
$ cat /usr/pruebas/archivo
Texto del documento uno
$ cat /usr/director/prueba
prueba de textos.
fichero de prueba
$ ls /usr/director
prueba
carta
$
```

Hemos representado un fichero por el camino que éste recorre dentro del árbol del sistema, desde la raíz del mismo (root). Este camino se denomina en UNIX **pathname** y su significado es el recorrido que se debe hacer en el árbol del sistema para localizar el fichero, partiendo desde la base (root) del árbol. Como vemos, se ha listado el contenido de un fichero "prueba" residente en el usuario "director". Es una norma universal de UNIX el poder representar un fichero por su camino o "pathname". Asimismo es posible listar los nombres de ficheros de otro usuario del sistema mediante su camino (pathname) del árbol. La estructura de árbol que reside en un sistema UNIX con varios usuarios podría ser la representada en la figura 1.

Vamos a ejecutar el comando **cp,** definiendo ficheros por su camino (pathname) en el árbol de directorios del sistema:

```
$ cp /usr/director/prueba datos
$ ed /usr/director/prueba
....
....
$
```

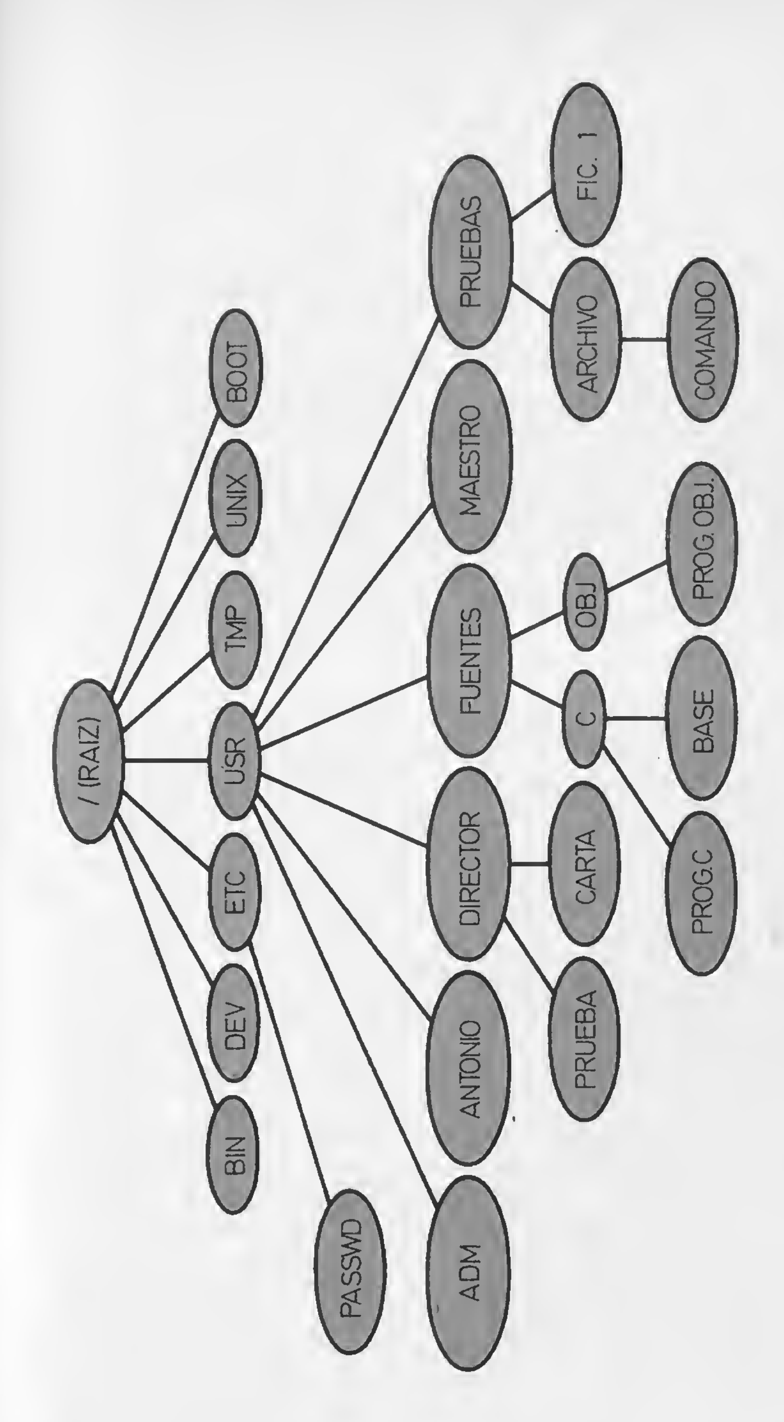



Fig. 1.—Posible estructura de árbol de un sistema UNIX con varios usuarios.

Como podemos ver, hemos efectuado la copia de un fichero del usuario "director" sobre el fichero "datos" en el directorio donde el usuario está trabajando (normalmente **/usr/pruebas).** Asimismo, hemos editado el propio fichero del usuario "director". Llegados a este punto vamos a desarrollar un tema importante en UNIX: los permisos de acceso a ficheros del sistema.

Permisos de acceso a ficheros

Cada uno de los ficheros residentes en el sistema posee un conjunto de permisos de acceso para los usuarios del mismo. El usuario puede cambiar los permisos de acceso de "sus" ficheros, de tal forma que proteja la información de alguna manera. Como comentamos en otro capítulo, existe un **súper usuario** del sistema, que puede volver a cambiarnos nuestros permisos, de tal forma que desproteja y tenga acceso a nuestra información.

El sistema reconoce a cada usuario por un cierto número que se le asigna a la hora de su creación. El nombre del usuario no es más que un identificador para el sistema y una especie de clave de acceso al mismo. Debido a esto varios nombres de usuarios distintos pueden tener el mismo número identificador. Esta situación no es nada segura dentro de un sistema y el administrador del mismo se encargará de controlarla. Cada usuario, además de su número de identificación, posee un número identificador del grupo al que pertenece. En muchos de los sistemas UNIX el grupo de todos los usuarios del sistema se denomina **other.** Toda la información referente a los parámetros de los grupos y usuarios del sistema se encuentra en un fichero localizado en el pathname **/etc/passwd.**

Existen tres tipos de permisos de acceso a un fichero del sistema

- read → **r** (lectura)
- write → **w** (escritura)
- execute → **x** (ejecución)

La opción **-l** del comando **ls** nos muestra los permisos de acceso que posee el fichero que se visualiza:

```
$ ls -l /etc/passwd
-rw-r--r--  1  root      2048  May 9   13:50  /etc/passwd
$ ls -lg /etc/passwd
-rw-r--r--  1  adm       2048  May 9   13:50  /etc/passwd
$ ls -l /bin/passwd
-rwxr-xr-x  1  root      8484  Feb 24  10:00  /bin/passwd
$ ls -l /bin/who
-rwxrwxr-x  1  root      6348  Mar 1   08:00  /bin/who
$ ls -l archivo
-rw-r--r--  1  pruebas     24  May 17  18:30  archivo
$
```


La opción **-l** del primer comando nos informa del nombre del propietario o usuario del fichero, que corresponde al nombre de usuario **root.** La opción **-g** nos muestra el nombre del grupo del fichero, en este caso **adm.** La cadena **-rw-r--r--** nos da información acerca de los permisos de los ficheros. El primer carácter de la cadena (-) indica el tipo de fichero de que se trata; en este caso, un fichero ordinario. En caso de ser un directorio, aparecería el carácter **"d".** Los siguientes tres caracteres de la cadena indican los accesos autorizados del fichero para el propio usuario del mismo. Las tres siguientes son para el **grupo** de usuarios al que pertenece el propio usuario creador del fichero y las tres últimas posiciones corresponden a los accesos para el resto de usuarios del sistema. Veamos un cuadro de interpretación de estos temas:

ACCESO A FICHEROS	
-	Indica que NO hay permiso para acceder al fichero.
r	Se permite acceso de lectura al fichero.
w	Se permite acceso de escritura al fichero.
x	Se permite la ejecución del fichero. Contiene un proceso ejecutable por el sistema.
Cadena de permisos de acceso "tabdefghi" posiciones 1234567890	
t → posición 1	tipo de fichero: - ordinario **d** directorio
a → posición 2	lectura para el usuario: - no permitida **r** permitida.
b → posición 3	escritura para usuario: - no permitida **w** permitida.
c → posición 4	ejecución para usuario: - no permitida **x** permitida.
d → posición 5	lectura para el grupo de usuarios.
e → posición 6	escritura para el grupo de usuarios.
f → posición 7	ejecución para el grupo de usuarios.
g → posición 8	lectura para el resto de usuarios.
h → posición 9	escritura para el resto de usuarios.
i → posición 0	ejecución para el resto de usuarios.

Ejemplos:

-rw-r--r-- → fichero ordinario. Permitida lectura y escritura al usuario del fichero. Permitida lectura al grupo y resto de usuarios. No permitida escritura al grupo y resto de usuarios. No permitida ejecución a ningún usuario del sistema.
-rwxr-xr-x → fichero ordinario. Permitida lectura y ejecución a todos los usuarios, usuario propio, grupo y resto de usuarios. Permitida escritura sólo al usuario del fichero.
-rwxrwxr-x → fichero ordinario. Permitida lectura y ejecución a todos los usuarios del sistema. Permitida escritura al usuario y al grupo. No permitida escritura al resto de usuarios.

Cuando nosotros enviamos a la Shell la orden de ejecución de un comando, por ejemplo **who,** ésta busca en una serie de directorios del sistema hasta encontrar dicho fichero. Uno de los directorios buscados es **/lib,** donde se encuentra el fichero **who.** El núcleo o **kernel** del sistema UNIX compueba los permisos de acceso al fichero y obra en consecuencia. Si este comando posee permiso de ejecución para el usuario que los desea ejecutar, el sistema validará la ejecución; en caso contrario no posee permiso de ejecución, nos avisará con un mensaje y no ejecutará el comando.

Los permisos de acceso a directorios tienen igual estructura, pero se diferencian en algún aspecto:

```
$ ls -ld .
drwxrwxr-x  3  pruebas        50 May 12 23:00 .
$
```

Como vemos, hemos incluido la opción **-d** dentro del comando **ls.** Esta opción nos visualiza la información de un directorio (la notación) "." corresponde a la nomenclatura del propio directorio del usuario). La protección **r** indica que el directorio tiene permiso de lectura. La protección **w** indica que se pueden crear y borrar ficheros dentro del directorio. El permiso para borrar un fichero dentro de un directorio es independiente del propio fichero. Si el usuario posee un permiso **w** en un directorio, puede borrar ficheros del mismo incluso si éstos están protegidos. El comando **rm** solicitará confirmación para borrar los ficheros protegidos. La protección **x** en un directorio indica que se posee acceso de búsqueda de ficheros dentro del directorio. Es decir, que

si un usuario posee permiso de ejecución en un directorio pueden buscar ficheros a través del mismo. Con estas premisas se puede asimilar las tareas que efectúan las protecciones de acceso a directorios.

Vamos a explicar, sin entrar en excesivo detalle y por medio de una tabla, el comando **chmod,** que efectúa el cambio de permisos de acceso en los ficheros:

COMANDO CHMOD	
chmod <quien> <operación> <permiso > <ficheros>	
<quien>	
u	usuario propio del fichero
g	grupo de usuarios asociados al usuario
o	resto de usuarios del sistema
a	todos (usuario, grupo y resto). Opción por defecto
<operacion>	
+	añade permiso digitado
-	quita permiso digitado
=	supone permiso absoluto para los ficheros
<permiso>	
r	permiso de lectura
w	permiso de escritura
x	permiso de ejecución

Ejemplos:

```
$ ls -l ejemplo
-rwxrwxrwx   .....
$ chmod go-wx ejemplo
$ ls -l ejemplo
-rwxr--r--   .....
$ chmod g+x ejemplo
$ ls -l ejemplo
-rwxr-xr-x   .....
$ chmod a=w ejemplo
$ ls -l ejemplo
-rwxrwxrwx   .....
$ chmod r-x ejemplo
$ ls -l ejemplo
-rwxrwxrw-   .....
```

Existe otra forma de manejo del comando, más complicada, que dejamos a gusto del lector el consultarla en los manuales originales de UNIX.

Vamos a desarrollar el comando **cd**, que permite el movimiento del usuario por los directorios propios o por los del sistema. El comando **cd** posiciona al usuario en el directorio indicado con una cierta nomenclatura:

cd	cambia al directorio normal del usuario: /usr/<nombre usuario>
cd <recorrido>	cambia al directorio especificado por <recorrido>, que debe ser un camino (pathname) completo.

Ejemplos:

```
$ cd
$ pwd
/usr/pruebas
$ cd /usr
$ pwd
/usr
$ cd /usr/pruebas/programas
$ pwd
/usr/pruebas/programas
```

cd <subdirectorio>	cambia al subdirectorio indicado. No hace falta poner el recorrido completo. Debe ser un directorio dentro del actual en el que se está.
cd ..	sube un nivel en la estructura (jerarquía) del árbol.
cd .	se mantiene en el directorio actual

Ejemplos:

```
$ cd
$ cd programas
$ pwd
/usr/pruebas/programas
$ cd ..
$ pwd
/usr/pruebas
$ cd archivo
archivo: bad directory
$ cd
$ cd .
$ pwd
/usr/pruebas
$
```


Veamos dos comandos que permiten la creación y borrado de directorios. El comando **mkdir** (creación de directorios) y el comando **rmdir** (borrado de directorios).

mkdir <nombre de directorios>

```
$ cd
$ mkdir sub
$ cd sub
$ pwd
/usr/pruebas/sub
$ mkdir fuentes textos
$ cd fuentes
$ pwd
/usr/pruebas/sub/fuentes
$ cd ..
$ pwd
/usr/pruebas/sub
$
```

El comando **mkdir** crea los directorios dentro del directorio actual de trabajo del usuario. Para crear un directorio se debe poseer permiso de escritura en el directorio donde se encuentre el usuario, es decir, en el directorio padre del que se va a crear.

```
$ cd
$ cd sub
$ ls -d
fuentes   textos
$ rmdir textos
$ ls -d
fuentes
$ cd .
$ pwd
/usr/pruebas
$ rmdir sub
rmdir: sub not empty
$
```

En caso de que un directorio contenga algún fichero o directorio, el comando **rmdir** no permitirá su borrado y nos presentará un mensaje de aviso. La eliminación de un directorio, al igual que la de un fichero, es aceptada por el comando si el usuario posee permiso de escritura en el mismo.

Búsqueda de ficheros en directorios

Existe un comando UNIX que permite la búsqueda o localización de un fichero dentro del árbol de directorios. Su nombre es **find**. Las especificaciones que se pueden dar para la búsqueda

del fichero incluyen posibilidades desde el nombre del fichero, propietario del mismo, nombre del grupo de pertenencia, enlaces (links) e, incluso, fecha-día de última modificación o acceso al mismo.

find <directorios> <condiciones búsqueda>

busca a partir de los directorios especificados los ficheros que cumplen las condiciones dadas.

La lista de directorios debe estar separada por blancos o tabuladores, y efectúa la búsqueda a partir de esos directorios y sus subdirectorios.

Condiciones:

-name <ficheros>	busca los ficheros especificados;
-type [df]	d → fichero tipo directorio; f → fichero totalmente lleno;
-links <n>	busca ficheros con <n> enlaces;
-user <usuario>	busca los ficheros correspondientes al <usuario> especificado;
-group <grupo>	busca los ficheros correspondientes al grupo especificado;
-size.<n>	busca ficheros con tamaño de <n> bloques (512 bytes);
-atime <n>	busca ficheros en los cuales hace <n> días se ha efectuado un acceso;
-mtime <n>	busca ficheros en los que hace <n> días se ha efectuado una modificación.
-print	imprime el camino de recorrido dentro del árbol de directorios del sistema.

CAPITULO V

EL INTERPRETE DE COMANDOS ("SHELL") DEL SISTEMA

Una vez realizada la conexión al ordenador por medio del comando "login", el sistema devuelve un "prompt" indicando que se encuentra preparado para recibir una orden desde el terminal. El carácter de "prompt" consiste en un signo "$" para usuarios normales o un "#" para el súper-usuario o "root".

Desde el punto de vista del usuario, el intérprete de comandos proporciona un medio sencillo de comunicarse con el sistema, ofreciéndole dos posibilidades:

a) Ejecutar directamente cualquier comando del ordenador.
b) Escribir y ejecutar instrucciones pertenecientes al propio lenguaje del intérprete de comandos.

Entre las características más importantes del intérprete de comandos, llamado "Shell" ("concha" o "coraza" que aísla y protege al usuario del sistema), se pueden citar:

- Ejecución directa de comandos del sistema operativo, traspaso de opciones y argumentos, y sustitución de variables y cadenas de caracteres.
- Construcciones de alto nivel como while, if-then-else, case y for.
- Modificación del entorno de ejecución ("environment") del usuario (conjunto de variables y modos de funcionamiento predefinidos).
- Reencaminamiento de la Entrada/Salida hacia ficheros, y comunicación de procesos a través de "pipes" o tuberías.

Comandos simples

Los comandos simples consisten en una o más palabras separadas por espacios en blanco. La primera palabra es el nombre del comando a ejecutar y las siguientes son las opciones y argumentos del comando. Por ejemplo,

date <RETURN>

ejecutaría el comando "date", devolviéndonos como resultado la fecha y hora del sistema en un formato como:

Thu May 15 21:15:05 GMT + 1:00 1986

Indicando que se trata del jueves 15 de mayo de 1986, son las 21 horas, 15 minutos y 5 segundos, con una diferencia de una hora (+1:00) con respecto al meridiano de Greenwich (GMT).

Análogamente, el comando

ls —l

produciría un listado del directorio (comando ls) en formato extendido (opción —l).

Para proceder a la ejecución de un comando, normalmente la Shell crea un nuevo proceso y espera hasta su finalización. Una línea como

cc nombre.c

llamaría al compilador de lenguaje C para proceder a compilar el programa fuente "nombre.c", devolviendo el prompt "$" de la Shell una vez finalizada la compilación del programa.

En cambio

cc nombre.c&

llevaría a cabo la compilación del programa en "background" (en modo no interactivo), de manera que se produciría la compilación del programa por medio de un nuevo proceso creado en ese momento, devolviéndose el control al terminal inmediatamente, sin esperar a que finalice la compilación.

El operador "&" indica a la Shell que no debe esperar la terminación de un proceso. Para permitir el control del mismo por medio de comandos como "ps" ("process.status", muestra el estado de uno o varios procesos) o "kill" (fuerza la terminación de un

proceso), la Shell devuelve su número de proceso o "pid" (process identifier).

Redireccionamiento de la entrada/salida

La casi totalidad de los comandos de UNIX envían su salida al fichero de salida estándar, correspondiente en principio al terminal, aunque esta correspondencia puede modificarse y lograr el reencaminamiento de la salida hacia cualquier otro fichero. Por ejemplo, la ejecución del comando

ls —l > fichero

no produciría ninguna salida sobre el terminal, enviando el resultado de la ejecución del comando ls —l (listado de un directorio en formato extendido) al fichero denominado "fichero".

El carácter ">" indica un redireccionamiento de la salida estándar. Si el fichero indicado para la salida no existiese será creado por la Shell mientras que si, por el contrario, ya existiese, vería reemplazado su contenido por la salida del comando; si el fichero no fuese accesible o no se pudiese crear se obtendría un mensaje de error.

Análogamente, se puede dirigir la entrada estándar por medio del carácter "<", utilizado como

wc < fichero

Ejecutando el comando "wc" ("word counter" o contador de palabras), que cuenta el número de líneas, palabras y caracteres presentes en el fichero de entrada estándar, en este caso "fichero", debido al redireccionamiento de la entrada.

El reencaminamiento de la salida estándar de un programa a la entrada estándar de otro, se logra mediante el empleo de los "pipes" ("|") o tuberías. Así:

ls | sort —r

produciría un listado del directorio clasificado por orden alfabético inverso: "ls" daría lugar a un listado del directorio sobre la salida estándar, que se vería redireccionada por efecto del "pipe" "|" hacia la entrada del comando "sort", que mediante la opción "—r" produciría la clasificación por orden alfabético inverso.

Es algo semejante al efecto de los comandos:

ls > auxiliar
sort —r auxiliar

con la diferencia de que no es necesaria la utilización del fichero intermedio "auxiliar". Además, los dos comandos conectados mediante un "pipe" se ejecutan simultáneamente y no uno a continuación de otro, como sucedería en este segundo caso.

Los "pipes" no son bidireccionales (para ello se requeriría la utilización de dos "pipes"), en el sentido de que no permiten la lectura y escritura en ambos sentidos, sino sólo la escritura en un extremo y la lectura en el otro, y realizan su sincronización por medio de la detención de los comandos cuando éstos no tienen nada que leer o escribir. Así, se detendría la ejecución del comando "sort" cuando no tuviese nada que leer y la de "ls" cuando no tuviese nada más que escribir.

En el sistema operativo UNIX se denominan "filtros" a los programas que actúan sobre la entrada estándar y la modifican de alguna manera antes de devolverla a la salida estándar. Un ejemplo típico es el del comando "grep", que muestra a su salida aquellas líneas de entrada que contengan una determinada palabra. La ejecución de

grep factura movimientos

provocaría que se listasen sobre el fichero de salida estándar todas las líneas de un fichero denominado "movimientos", que contuviesen en algún lugar la palabra "factura", actuando como un "filtro" de las líneas de entrada.

Un "pipe" puede constar de más de dos comandos, produciéndose el reencaminamiento múltiple de sus entradas y salidas consecutivamente. Por ejemplo,

ls | grep carta | more

Producirá como resultado un listado por pantalla con paginación (espera antes de pasar a una nueva página) de todos los nombres de ficheros que contengan en algún lugar la palabra "carta".

Para finalizar realicemos un pequeño resumen acerca de la redirección de la entrada y salida:

- > f1 La salida estándar (descriptor de fichero 1) se envía al fichero "f1", creándose si no existiese anteriormente.
- >>f1 La salida estándar se envía al fichero "f1". Si ya existía el fichero entonces se añade al contenido anterior; en caso contrario se crea el fichero.
- <f1 Se toma la entrada estándar (descriptor de fichero 0) a partir del fichero "f1".
- <<f1 Se toma la entrada estándar a partir de líneas de entrada de la Shell hasta que se encuentre una línea que contenga la palabra "f1".

- >& num El descriptor de fichero "num" se duplica por medio de la función primitiva "dup", utilizándose el resultado como fichero de salida estándar.
- <& num La entrada estándar se duplica a partir del descriptor de fichero "num".

Generación de nombres de ficheros

La Shell proporciona un mecanismo para obtener nombres de ficheros con arreglo a una máscara de formato determinada. Por ejemplo,

ls —l *.c

Produce un listado de todos los nombres de ficheros que posean la terminación ".c" (habitualmente programas fuente en lenguaje C). El carácter "*" consiste en un formato que verifica cualquier carácter, incluso el carácter nulo. Las máscaras usadas son:

*	Verificado por cualquier cadena de caracteres, incluso el carácter nulo.
?	Verificado por un único carácter.
[...]	Verificado por cualquiera de los caracteres encerrados entre los paréntesis cuadrados ([ ]). Si aparecen un par de caracteres separados por un guión "—", entonces se refiere al rango de caracteres comprendido entre los dos especificados.

Por ejemplo

ls bloque?

produciría un listado del directorio con todos los nombres de ficheros que comenzasen por la palabra "bloque" y contuviesen a continuación cualquier carácter (bloque1, bloque9, bloquez, ...).

[a—m]*

Sería verificado por todos los nombres de ficheros que comenzasen por una letra comprendida entre la "a" y la "m", inclusives.

El carácter "*" no incluye aquellos nombres de ficheros que comiencen por un punto ".". Así, si contuviésemos un fichero con

el nombre ".profile", por ejemplo, no sería mostrado mediante el comando

ls *

Pero sí lo sería mediante

ls .*

Caracteres especiales (metacaracteres)

Los caracteres que poseen un significado especial para la Shell, como ">", "<", "*", "?", "|" y "&" se denominan metacaracteres. Cualquier carácter precedido por el signo " \ " se interpreta también como un metacarácter, cambiando su significado. Así:

echo \?

mostrará en pantalla un único carácter "?", mientras que

echo \\

hará lo mismo con el carácter " \ ".

Con el fin de permitir que las cadenas de comandos puedan alcanzar la longitud suficiente, con todas sus opciones, se ignora el carácter de salto de línea " \n" en medio de una cadena de caracteres.

Para proteger una cadena de caracteres de la Shell y evitar que pueda ser interpretada como si contuviese metacaracteres, se puede rodear la cadena por comillas simples " ", como en el caso de:

cat "bloque?"

Que produciría un listado por pantalla (comando cat) del fichero "bloque?", en lugar de referirse a los nombres de ficheros "bloque0", "bloque1", ... "bloquez".

Programación en el lenguaje de la Shell

La Shelll puede emplearse para leer y ejecutar comandos contenidos en un fichero. Por ejemplo, la instrucción

sh fichero [arg1 arg2 ... argn]

produciría una llamada a la Shell para que leyese y ejecutase los comandos contenidos en "fichero", diciéndose entonces que "fichero" es un "fichero de comandos".

Se pueden utilizar como argumentos en la llamada al fichero de comandos, los "parámetros posicionales" $1, $2, ... $9. Por ejemplo, si el fichero f1 contiene el texto

cc —o $1 $1.c

entonces

sh f1 nombre

es equivalente a

cc —o nombre nombre.c

Donde vemos que el parámetro posicional $1 ha sido reemplazado por "nombre". Este ejemplo serviría para compilar un programa fuente en lenguaje C llamado "nombre.c", dejando el programa compilado en "nombre", en lugar de "a.out", como denomina por defecto el compilador de C a su salida.

También puede convertirse un fichero de comandos en "ejecutable" con sólo cambiar sus protecciones de grupo y de usuario. Así

chmod +x f1

produce el efecto de cambiar la protección de ejecución del fichero f1 para el usuario convirtiéndolo en ejecutable, de manera que

f1 nombre

es equivalente a

sh f1 nombre

permitiendo que pueda intercambiarse el empleo de ficheros de comandos de la Shell y auténticos comandos del sistema operativo. La ejecución del fichero de comandos f1 sería equivalente a la ejecución directa de cualquier comando, creándose en cada caso un nuevo proceso para proceder a su ejecución.

La Shell proporciona dos variables reservadas para su empleo con parámetros posicionales, como son el número de argumentos de llamada, contenido en la variable **"$#"**, y el nombre del

fichero en ejecución, o **"$0"**. También existe una variable reservada **"$*"**, equivalente a todos los parámetros posicionales con excepción del primero **("$0")**.

Sentencias de control en la Shell

Es relativamente frecuente la ejecución de la Shell en bucles que incluyan parámetros posicionales $1, $2, ... y se ejecuten una vez por cada parámetro posicional. Por ejemplo, pensemos en un fichero de comandos denominado "tel" que busque nombres en un fichero "/usr/agenda/teléfonos", conteniendo líneas como

```
"..."
juan 4731291
tomas 4167801
fernando 4353218
alfredo 4450153
"..."
```

Si el fichero de comandos "tel" contuviese

```
for i
do
        grep $i /usr/agenda/telefonos
done
```

La línea de comando

tel juan

mostraría en pantalla aquellas líneas que contuviesen la cadena de caracteres "juan", en el fichero "/usr/agenda/teléfonos".

Del mismo modo

tel juan fernando

imprimiría las líneas que contuviesen el nombre "juan", y a continuación las que contuviesen el nombre "fernando".

En general, el bucle for se presenta como:

```
for var in v1 v2 ..
do
        lista-de-comandos
done
```

Donde lista-de-comandos se refiere a uno o más comandos simples separados por un salto de línea o por un punto y coma

";". "do" y "done" son dos separadores para delimitar el cuerpo del bucle; "var" es una variable de la Shell y "v1", "v2", son algunos de sus posibles valores.

Cada vez que se encuentre un valor de "var" comprendido entre "v1", "v2", ..., se ejecutará el conjunto de instrucciones comprendidas en el cuerpo del bucle for.

En el caso de que "v1", "v2", ... se omitan, entonces se ejecutará el bucle para cada uno de los parámetros posicionales, como si se tratase de la variable "$*".

La sentencia case

El comando "case" tiene la forma general

```
case var in
     v1) comando;;
     v2) comando;;
     "..."
esac
```

La Shell comprueba secuencialmente si el valor de "var" coincide con "v1", "v2", ..., y en caso de que coincida ejecuta el comando(s) correspondiente(s) finalizando la ejecución del "case". Como el "*" es un valor que verifican todas las variables, se puede utilizar para implementar un valor por defecto, incluyéndolo justo antes de la sentencia "esac", indicativa del final del bucle case. Por ejemplo,

```
case $# in
        1) cat >> $1;;
        2) cat >> $2 < $1;;
        *) echo  empleo: añadir [desde] a ;;
```

sería un comando para añadir (sumar) un fichero a otro. Utilizado como

suma f1

$# valdría 1, copiando la entrada estándar al final del fichero f1 por medio del comando cat. Análogamente,

suma f1 f2

añade el contenido del fichero f1 al final del fichero f2. Por el contrario, si el número de argumentos de llamada fuese distinto de 1

ó de 2, entonces se alcanzaría el valor por defecto "*", pasando a imprimirse un mensaje de error.

También pueden emplearse valores alternativos en cada uno de los "v1", "v2"..., por medio del empleo de una barra vertical "|". Así:

```
case $x in
        -S|-s) ...
        -N|-n) ...
esac
```

sería equivalente a

```
case $x in -[Ss]) ...  -[Nn]) ...  esac
```

Variables de la Shell

La Shell maneja variables del tipo cadena de caracteres. Su nombre consiste en una combinación de letras (forzosamente el primer carácter), dígitos y caracteres "_". La forma de declarar las variables es asignándoles un valor, como en

```
i=100 x1_max=f200j s1=f1
```

que asignaría a las variables "i", "x1_max" y "s1" los valores "100", "f200j" y "f1", respectivamente.

Para acceder al valor de una variable se antepone un carácter "&" a su nombre. Por ejemplo:

echo i

imprimiría literalmente el carácter "i", mientras que

echo $i

imprimiría el valor de la variable "i", en este caso 100.

Es frecuente el empleo de variables de la shell para almacenar nombres de directorios, facilitando su empleo. Así

```
d=/usr/juan/programas/fuentes/c
mv *.c $d
mv /etc/*.c $d
```

llevaría los ficheros fuentes en lenguaje C contenidos en el directorio actual de trabajo, y en el directorio /etc, al directorio "/usr/juan/programas/fuentes/c".

También se emplea la construcción

echo $ {user}

equivalente en este caso a

echo $user

pero que se utiliza cuando el nombre de la variable viene seguido por una letra o dígito, como es el caso de

```
tmp=/tmp/pruebas
ps a >${tmp}1
```

produciría el reencaminamiento de la salida del comando ps al fichero "/tmp/pruebas1", mientras que

```
ps a >$tmp1
```

Se referiría al valor de la variable "tmp1", completamente distinto al anterior.

Las siguientes variables son inicializadas por la shell al proceder a la ejecución de cada comando:

- $? Consiste en el código de retorno (status) del último comando ejecutado, devuelto como una cadena de caracteres decimales. Normalmente se emplea el valor 0 tras la ejecución de un comando satisfactoriamente y un valor distinto de cero para indicar la existencia de algún problema en su ejecución.
- $# Es el número de parámetros posicionales, en decimal.
- $$ Consiste en el número identificador de proceso de esta shell, en decimal. Se utiliza con frecuencia para generar nombres de ficheros temporales, con la garantía de que sean únicos. Por ejemplo,

```
ps a >/tmp/pruebas$$
"..."
rm /tmp/pruebas$$
```

- $! Es el número identificador de proceso del último comando ejecutado en modo "background" (no interactivo).

Algunas variables poseen un significado especial para la shell, pasándose al "environment" o entorno de ejecución del usuario, de modo que son accesibles desde otros programas.

- $MAIL Cuando se maneja interactivamente, la shell busca esta variable antes de devolver su prompt "$". Si el fichero especificado en la variable ha sido modificado desde la última consulta la shell imprimirá el mensaje "You have mail" antes de pasar a solicitar el siguiente comando. Normalmente, MAIL se inicializa en el fichero ".profile", existente en el directorio de trabajo del usuario. Por ejemplo,

  MAIL=/usr/spool/mail/juan

  Inicializaría MAIL al valor de un nombre de fichero en el que almacenar el correo manejado por el comando "mail".
- $HOME Es el argumento por defecto para el comando "cd" (cambio de directorio). Así la utilización de "cd" sin argumentos es equivalente al empleo de

  cd $HOME

  La variable HOME también suele inicializarse en el fichero de comandos ".profile".
- $PATH Consiste en una lista de directorios en los cuales deben de buscarse los comandos a ejecutar. Cada vez que se procede a la ejecución de un comando la shell busca este comando en los directorios (y en el orden) especificados por PATH o, por defecto, en "/bin" y "/usr/bin". PATH suele inicializarse en el fichero ".profile", consistiendo en una serie de nombres separados por el carácter dos puntos ":"

  PATH=:/usr/juan/bin:/usr/ucb:/bin:/usr:bin
- $PS1 es el prompt primario de la shell, por defecto el carácter "$".
- $PS2 es el prompt secundario de la shell, cuando la introducción de un comando ha sido incompleta y debe proseguir la entrada. Por defecto consiste en el carácter "<".

El comando test

El comando "test", aunque no forma parte de la shell, se utiliza para su uso exclusivo. Consiste en la evaluación de una determinada condición, devolviendo un valor cierto o falso según que se cumpla o no. Por ejemplo:

test —f f1

Devuelve el valor 0 si el fichero f1 existe y un valor distinto de cero en caso contrario. Entre las comprobaciones más habituales se pueden citar:

test s		cierto si la cadena "s" no es nula.
test —f	fichero	cierto si "fichero" existe.
test —r	fichero	cierto si "fichero" se puede leer.
test —w	fichero	cierto si "fichero" se puede escribir.
test —d	fichero	cierto si "fichero" es un directorio.

Sentencias while y until

El bucle "while" tiene la forma general:

```
while condición
do
    comando(s);
done
```

Donde la condición comprobada por el while se suele incluir la ejecución de un comando "test". Cada vez que se comprueba la condición y ésta es cierta se ejecuta el comando(s) incluido(s) en el cuerpo del bucle. Por ejemplo,

```
while test $1
do
    "..."
    shift
done
```

Es equivalente a la construcción

```
for i
do
    "..."
done
```

"shift" es un nuevo comando de la shell que desplaza los parámetros posicionales $2, $3, ..., convirtiéndolos en $1, $2, ..., y perdiendo el parámetro $1 anterior.

El bucle "until" es semejante al "while" con la diferencia de que se invierte la condición de permanencia en el bucle: se ejecuta el bucle mientras la condición sea falsa y finaliza la ejecución cuando sea cierta.

La sentencia if

La shell posee una sentencia condicional if-then-else, de la forma:

```
if condición
then
    comando
else
    comando
fi
```

La condición lógica incluida en el "if" suele consistir en una instrucción "test", al igual que anteriormente en el caso del "while". Las alternativas "then" y "else" se refieren a las acciones a ejecutar en el caso de que la condición lógica comprobada en el "if" sea cierta o falsa, respectivamente.

La partícula "fi" indica el final del rango de la construcción "if". La secuencia

```
if comando1
then comando2
fi
```

Puede escribirse también como

```
comando1 && comando2
```

Del mismo modo

```
comando1 || comando2
```

ejecutaría "comando2" solamente si "comando1" devuelve un valor falso. En cada uno de los casos el valor devuelto por la expresión corresponde al del último comando simple ejecutado.

El comando man

El siguiente ejemplo corresponde al comando "man" de UNIX, utilizado para imprimir secciones del manual de UNIX. La manera de utilizarlo es, por ejemplo:

```
man sh
man 2 exec
```


En el primer caso se imprimiría la sección de manual correspondiente a la shell, comenzando por la primera sección al no especificarse ninguna. El segundo ejemplo imprimiría la página de manual correspondiente al comando "exec" en la sección 2.

A continuación se muestra una versión del comando "man", realizado íntegramente en el lenguaje de comandos de la shell:

```
: dos puntos ":" representa un comentario.
: los valores por defecto son nroff ($N),
: y sección 1 ($s)
N=n s=1

for i
do case $i in
        [1-9]*) s=$i;;

        -t) N=t;;

        -n) N=n;;

        -*) echo opción desconocida \ $i\ ;;

        *) if test -f man$s/$i.$s
                then $[N]roff man0/${N}aa man$s/$i.$s
                else : busca a través de todas las secciones
                        found=no

                        for j in 1 2 3 4 5 6 7 8 9
                        do if test -f man$j/$i.$j
                                then man $j $i
                                        found=yes
                                fi
                        done

                        case $found in
                                no) echo  no existe página $i
                        esac
                fi
        esac
done
```

El programa utilizaría los impresores "troff" o "nroff", tomando este último por defecto, al inicializar N=n. Si entre los argumentos aparece un número en el rango de 1 a 9 este número se toma como la sección. Tras comprobar la existencia del comando y la sección especificados, procede a su impresión, devolviendo un mensaje de error en caso contrario.

Sustitución de parámetros

Si un parámetro de la shell no ha sido inicializado explícitamente, entonces se le asigna el valor nulo. Por ejemplo, si la variable "d" no ha sido inicializada:

echo $d

o

echo $ {d}

no harán nada, al ser "d" una cadena de caracteres nula. Sin embargo, es posible especificar un valor por defecto en la forma:

echo $ {d—.}

que hace que se imprima el valor de "d" si esta variable ha sido inicializada, o un punto "." en caso contrario. Análogamente,

echo $ {d—$1}

mostrará el valor de "d" si ha sido inicializada, o el valor del parámetro posicional $1 en caso contrario.

También se emplea con frecuencia la notación

echo $ {d?mensaje}

que mostrará el valor de "d" o el texto "mensaje", finalizando la ejecución del programa. Si no se ha especificado "mensaje", entonces se emplearía un mensaje de error. Un programa shell que requiera algunos parámetros podría entonces inicializarse como:

:$ {usuario?} $ {acct?} $ {bin?}

Donde el carácter dos puntos ":" es un comando que no hace nada una vez que sus argumentos han sido evaluados. Si cualquiera de las variables "usuario", "acct" o "bin" no han sido inicializadas, entonces se abandonará la ejecución del programa.

Sustitución de comandos

Es posible sustituir la salida estándar de un programa de una manera parecida a como actúan los parámetros. El comando "pwd" ("print working directory", muestra el directorio de trabajo), imprime en la salida estándar el nombre del directorio actual. Por ejemplo, si estuviésemos en el directorio "/usr/juan/programas" y se ejecutase la instrucción

d='pwd'

sería equivalente a hacer

d=/usr/juan/programas

La totalidad de la cadena de caracteres comprendida entre comillas simples (' ') se considera como un comando a ejecutar y se reemplaza el texto entre (' ') por la salida de ese comando.

La sustitución de comandos tiene lugar en todos aquellos casos donde se realiza la sustitución de parámetros, permitiendo el empleo de comandos de procesamiento de cadenas de caracteres incluidos en instrucciones de la shell. Por ejemplo, consideremos el comando "basename" que elimina un sufijo (una terminación) determinada de una cadena de caracteres. Así:

basename main.c .c

Produciría como resultado la cadena "main" al eliminar la terminación ".c". Esto podría aparecer como una parte de una shell destinada a compilar programas C, en la que se incluiría una parte como:

```
case $A in
        "..."
        *.c)    B='basename $a .c'
        "..."
esac
```

inicializando B como la parte correspondiente al nombre $A, con la terminación ".c" suprimida.

Orden de evaluación

La shell es un procesador de macros que realiza sustitución de parámetros y comandos, así como generación de nombres de ficheros. Los comandos se analizan con arreglo al siguiente orden de prioridad y realización de sustituciones:

- Sustitución de parámetros. Por ejemplo $usuario.
- Sustitución de comandos. Por ejemplo 'pwd'. Sólo tiene lugar una evaluación, de manera que si, por ejemplo, el valor de la variable X es la cadena $y, entonces

  echo $X

  mostraría literalmente el texto "$y".

- Interpretación de espacios en blanco. De acuerdo con las sustituciones anteriores, los caracteres resultantes se parten en palabras distintas de blancos. Para esta finalidad se consideran "blancos" los caracteres de la cadena $IFS, variable de la shell (lo mismo que HOME y PATH, definidas anteriormente). Por defecto, la cadena de caracteres $IFS contiene un espacio en blanco, un tabulador y un salto de línea.
  La cadena nula no se considera como una palabra, a menos que vaya entre comillas simples (''). Así:

  ```
  echo ''
  ```

  tomaría un nulo como primer argumento del comando echo, mientras que

  ```
  echo $nulo
  ```

  ejecutará echo sin argumentos si la variable "nulo" no ha sido inicializada o su contenido es una cadena nula.
- Generación de nombres de ficheros. Se explora cada palabra buscando los caracteres "*", "?" y [...] y se genera una lista alfabética de nombres de ficheros para reemplazar la palabra. Cada nombre de fichero es un argumento por separado.

Esta lista de sustituciones que acaba de describirse tiene lugar para cada uno de los argumentos integrantes de un bucle "for".

Tratamiento de errores

El tratamiento de los errores detectados por la shell depende del tipo de error y de si la shell está siendo utilizada o no interactivamente. Una shell interactiva es aquella cuya entrada y salida está dirigida al terminal del usuario.

La ejecución de un comando puede producir un error por cualquiera de las siguientes razones:

- Fallos en la redirección de la Entrada/Salida. Por ejemplo, si un fichero no existe, no puede crearse, o no se puede abrir.
- El propio comando no existe o no puede ser ejecutado.
- El comando no termina normalmente su ejecución debido, por ejemplo, a un error de software o hardware.

- El comando termina su ejecución normalmente pero, sin embargo, devuelve un valor distinto de cero a su finalización.
- Errores de sintaxis. Por ejemplo if ... then ... done
- Una señal software que genera una interrupción.
- Fallo de algún comando embebido, como "cd".

El indicador "—e" de la shell causa su terminación en caso de que se detecte algún error durante la ejecución.

El cuadro siguiente muestra los valores de las señales software en UNIX:

1		entrada en bucle
2		interrupción
3	*	salir
4	*	instrucción ilegal
5	*	interrupción por ejecución "paso a paso"
6	*	instrucción IOT ("trap" de Entrada/Salida)
7	*	instrucción EMT
8	*	error en operación en coma flotante
9		finalizar (no puede ser detectada o ignorada)
10	*	error de bus
11	*	violación de segmentación
12	*	argumento o llamada al sistema erróneos
13		escribir en un pipe sin ningún proceso que lo lea
14		señal de alarma de reloj
15		terminación software

Las señales indicadas con un "*" producen un volcado de memoria si no son invalidadas. Las señales software que poseen algún interés para nosotros son las números 1, 2, 3, 14 y 15.

Los programas en shell normalmente terminan su ejecución cuando reciben una señal de interrupción desde el terminal. Se utiliza el comando "trap" cuando se desea ejecutar alguna acción antes de finalizar la ejecución, como borrar algún fichero temporal, cerrar ficheros, etc... Por ejemplo:

```
trap 'rm /tmp/ps$$; exit'2
```

inicializa un "trap" (subrutina de tratamiento de errores) para la señal número 2 (interrupción), de manera que cuando se reciba la señal se ejecutarán los comandos

```
rm /tmp/ps$$; exit
```

"exit" es otro comando de la shell que finaliza la ejecución de un programa. Es obligatorio incluirlo en este caso porque, si no, después de ejecutar el "trap" la shell proseguiría la ejecución a partir de la línea en la que se produjo la interrupción.

Las señales de UNIX pueden ser tratadas de tres maneras distintas: pueden ser ignoradas, en cuyo caso las señales nunca llegarán a enviarse al proceso; pueden ser detectadas, de modo que el proceso decida qué acción tomar cuando reciba una señal y, finalmente, se puede dejar que causen la terminación de un proceso sin por ello llevar a cabo ninguna acción posterior.

El listado siguiente muestra el programa "busca", que es un ejemplo de un "trap" sin sentencia "exit"; explora cada directorio incluido en el directorio de trabajo, pregunta su nombre y ejecuta los comandos introducidos desde el terminal hasta que se alcance un final de fichero o se reciba una interrupción. Las interrupciones se ignoran mientras se están ejecutando los comandos indicados, pero causan la terminación del programa cuando "busca" está esperando una entrada de datos por pantalla.

```
d='pwd'
for i in *
do if test -d $d/$i
        then cd $d/$i
                while echo "$i:"
                        trap exit 2
                        read x
                do trap : 2; eval $x; done
        fi
done
```

"read x" es un comando de la shell que acepta una línea a partir de la entrada estándar y deja el resultado en la variable "x". Devuelve un valor distinto tanto si se alcanza un final de fichero como si se produce una interrupción.

Ejecución de comandos

Para ejecutar un comando distinto de los incorporados por ella misma, la shell crea un nuevo proceso por medio de la función primitiva "fork" El entorno de ejecución para el comando incluye la entrada y salida estándar y el estado de las señales, estableciéndose en el proceso hijo antes de que se llegue a ejecutar el comando. También se utiliza "exec" en algunos casos cuando no se desea realizar un "fork", de modo que simplemente se reemplaza la shell por un nuevo comando. Por ejemplo, una posible versión del comando "nohup" (ejecuta comandos inmunes a las señales de UNIX) sería:


```
trap '' 1 2 3 15
exec $*
```

El "trap" invalida las señales 1, 2, 3 y 15, que seguirán siendo ignoradas por los comandos que se ejecuten posteriormente, al reemplazar "exec" la shell por cada uno de los comandos que figuren en "$*".

CAPITULO VI

EL NUCLEO DEL SISTEMA UNIX

Este capítulo describe en términos de alto nivel la implementación del núcleo del UNIX. La discusión está dividida en dos partes: la primera describe cómo ve el sistema UNIX los procesos, usuarios y programas, en tanto la segunda describe el sistema de E/S.

El núcleo de UNIX consiste en unas 10.000 líneas del código C y unas 1.000 de código ensamblador. El código ensamblador puede dividirse en unas 200 líneas, incluidas por motivos de eficiencia (podrían haber sido escritas en C) y 800 líneas para ejecutar funciones hardware que no son posibles hacer en C. Este código representa del 5 al 10 por 100 de lo que implica la extensa expresión "Sistema Operativo UNIX".

El núcleo es el único código de UNIX que no puede ser sustituido por un usuario a su antojo. Cómo se implementa el núcleo representa una gran responsabilidad y un gran poder. Todas las decisiones importantes deben ser ponderadas cuidadosamente. En todas partes la simplicidad ha sido sustituida por eficiencia.

Control de procesos

En UNIX varios procesos pueden ejecutarse a la vez. Por ejemplo, se podría ejecutar el programa sort en "blackground" y editar otro fichero en "foreground", mientras el sort se está ejecutando. Los procesos que se controlan directamente desde la pantalla se llaman procesos "foreground". Otros procesos que se han inicializado, pero que no se posee control sobre ellos, se llaman

procesos "background". A la vez sólo se puede tener un proceso foreground, pero múltiples procesos background pueden ejecutarse simultáneamente. Controlar los procesos foreground y background es el objeto de esta sección.

Ejecución y cancelación de un proceso en background

Para lanzar ún proceso en background se debe digitar un "&" al final del comando. Al perder el control sobre un proceso en background para abortarlo no se puede utilizar INTERRUPT; se tiene que utilizar el comando **kill**.

Para cancelar todos los procesos de background teclear:

$ kill 0

Para cancelar un solo proceso, teclear primero

$ ps

El comando **ps** muestra el número de identificación (PIDs) de todos los procesos activos para ese terminal.

Por ejemplo:

```
$ ps
 PID     TTY     TIME     CMD
3459      03     0:15     -sh
4831      03     1:52     cc program.s
5185      03     0:00     ps
```

En el ejemplo de arriba se podría teclear:

$ kill 4831

donde 4831 es el PID del proceso que se quiere cancelar.

El comando **ps** (estatus del proceso) se emplea para visualizar información sobre los procesos activos. Su sintaxis es:

$ ps [alx] lista-nombres

Opciones:

- a — Muestra todos los procesos asociados con un terminal.
- l — Formato extendido.

- x — Muestra todos los procesos no asociados con un terminal.

El formato extendido se compone de:

- F — flags asociados con los procesos. Valores de 01, 02, 04, 10 y 20
- S — Estado de los procesos
  O no existente
  S proceso durmiente (sleeping)
  W esperando
  R ejecutando
  I intermedio
  Z terminado
  T parado
- UID — El ID de usuario del propietario del proceso, siendo ID el número identificador.
- PID — El número del ID del proceso.
- PPID — El número del ID para los procesos padres.
- CPU — Utilización del proceso para scheduling.
- PRI — Prioridad de los procesos, números altos son prioridades bajas.
- NICE — Número usado en computación de prioridad.
- ADDR — Si reside en memoria, la dirección del núcleo; en otro caso la dirección del disco.
- WCHAN — El suceso por lo que el proceso está esperando o durmiendo. Si es un blanco, el proceso está en ejecución.
- TTY — El terminal en el que se ejecutan los procesos.
- TIME — El tiempo transcurrido de ejecución de los procesos.

Desarrollo de pipes en UNIX

Un pipe es un fichero ficticio que un programa podría crear y usar para pasar información a otros programas.

Los pipes se usan principalmente para pasar información entre programas, como el símbolo de la shell (|), en el que la salida de un programa es la entrada de otro. Esto elimina la necesidad

de crear ficheros temporales para pasar información a otros programas.

La librería estándar prevé varias funciones de manejo de pipes. Las funciones **popen** y **pclose** controlan a ambos, un pipe y un proceso. La función **popen** abre un pipe y crea un nuevo proceso a la vez. La función **pclose** cierra el pipe y espera la terminación de los procesos correspondientes. La función **pipe,** por otra parte, da acceso a bajo nivel a un pipe.

La función **popen** crea un nuevo proceso y abre un pipe al fichero estándar de entrada o salida de ese nuevo proceso. La función tiene la forma:

popen (comando, tipo)

Donde **comando** es un puntero a una cadena de caracteres que contiene un comando shell y **tipo** es un puntero a una cadena que define si el pipe va a ser abierto para lectura **"r"** o escritura **"w"**. Esta función normalmente devolverá el puntero al pipe abierto y NULL si se encuentra un error. Veamos un ejemplo:

```
FILE * pstrm;
pstrm = popen ("cat > response", "w");
```

La función **pclose** cierra el pipe abierto por la función **popen.** La función tiene la forma:

pclose (apuntador)

Donde apuntador es un puntero al fichero del pipe a cerrar. Veamos un ejemplo del tema:

```
FILE * pstrm;
pclose (pstrm);
```

La función **pipe** abre un pipe tanto para lectura como para escritura. La función tiene la forma:

pipe(fd)

Donde **fd** es un puntero a una matriz de dos elementos de tipo entero. Cada elemento recibe un descriptor de fichero, el primero para lectura y el segundo para escritura. La función normalmente devuelve 0, y -1 si se encuentra algún error. Veamos un ejemplo:

```
int chan[2]
```



```
if (pipe(chan) == -1)
    exit(2);
```

La función close se emplea para cerrar el pipe de lectura o escritura. Un ejemplo es:

```
close (chan[0]) cierra el pipe de lectura
close (chan[1]) cierra el de escritura.
```

APENDICE

COMANDOS DEL SISTEMA OPERATIVO

AR — **Manejo de archivos y librerías de ficheros**
ar [abclsv] [dmpqrtx] [nombre posterior] nombre del archivo fichero

a	añade nuevos ficheros tras el nombre especificado
b	añade nuevos ficheros antes del nombre especificado
c	suprime el mensaje al crear el fichero
l	sitúa temporalmente los ficheros en el directorio local
v	amplía la información dada por otras opciones
d	borra ficheros del archivo
m	mueve ficheros hasta el final del archivo
p	lista los nombres de los ficheros en el archivo
r	reemplaza ficheros en archivo
s	fuerza la regeneración de la tabla de símbolos del archivo
t	lista la tabla de contenidos del fichero de archivo
x	extrae ficheros desde el archivo

nombre posterior	posiciones del nombre de los ficheros; usualmente empleado con **a, b, i, r** y **m**
nombre del archivo	nombre del fichero de archivo
fichero...	uno o más ficheros a poner en el archivo

AS — **ensamblador**
as [-o objfile] **[-n] [-m] [-R] [-r] [-v] nombre del fichero**

—n	pone/quita direcciones largas/cortas

—m	ejecuta el pre-procesador de macros m4 a la entrada del ensamblador
—R	borra el fichero de entrada al finalizar el ensamblado
—r	pone todos los datos ensamblados en la sección **.text**
—v	escribe el número de versión del ensamblador sobre el fichero de errores estándar
nombre	nombre del fichero a ensamblar

AT —**ejecuta comandos en la fecha y hora especificadas**
at hora [día] [fichero]

hora [APNM]	especificada la hora y minuto; **A** indica **AM**, **P** indica **PM**, **N** indica tarde y **M** indica medianoche
día	se refiere tanto a un nombre de mes seguido por un número de día, u, opcionalmente, un día de la semana
fichero	fichero a emplear posteriormente como entrada para la **shell**

AWK — **reconocer de modelos y lenguaje de procesamiento**
awk [-Fc] [modelo acción] [fichero]

—Fc	emplea el carácter **c** como separador de campos
modelo	conjunto de modelos a reconocer
acción	acción a realizar cuando se encuentre el modelo
fichero	uno o más ficheros a buscar

BASENAME, DIRNAME — aísla partes de nombres de ficheros

basename	cadena de caracteres [sufijo]
dirname	cadena

cadena	indica cadena de caracteres a buscar
sufijo	indica sufijo (terminación) a eliminar del nombre.

BC — **lenguaje aritmético de precisión arbitraria**
bc [–c] [—l] [archivo...]

—c	sólo compila
—l	carga la librería matemática de precisión arbitraria
archivo	uno o más nombres de ficheros

BDIFF — **compara dos ficheros e informa de sus diferencias**
bdiff archivo1 archivo2 **[—n] [—s]**

archivo1	primer nombre de fichero
archivo2	segundo nombre de fichero
—n	compara un número (n) de líneas
—s	suprime diagnósticos

CAL — **imprime calendario**
cal [mes] año

mes	número de dos dígitos correspondiente al mes
año	número de cuatro dígitos correspondiente al año

CAT — **concatena e imprime ficheros**
cat [—e] [—n] [—s] [—t] [—u] [—v] fichero...

—e	pone el carácter **$** al final de la línea cuando se emplea con la opción **—v**
—s	omite mensajes de error sobre ficheros no existentes
—t	pone el carácter I como tabulador cuando se emplea conjuntamente con la opción **—v**
—u	causa que la salida sea sin buffer (carácter a carácter)
—v	hace que se muestren los caracteres no imprimibles
fichero	uno o más nombres de ficheros

CB — **copia un programa C poniendo espaciado e indentación.**
cb [—s] [—j] [—l longitud] [fichero]

—s	convierte el código al estilo propuesto en **Lenguaje de programación C** por **Kernighan y Ritchie**
—j	reúne las líneas partidas
—l longitud	longitud, parte las líneas más langas que **longitud**
fichero	uno o más ficheros de programas en C para reformatear

CC — **Compilador de C.**
cc [—c] [—Bserie] **[—E] [—O(KS)]** [—O volumen] **[—P] [—p] [—S] [—t[p012al]] [—v] [—voc,**arg1[,arg2...]] fichero...

—Bserie	construye nombres de ficheros para sustituir las fases del preprocesador, compilador, ensamblador y link-editor
—c	suprime la fase de link-edición
—E	ejecuta solamente **cpp** sobre los programas C indicados

—O(KS)	optimiza el código objeto K — optimizaciones dependientes del kernel (núcleo) S — optimiza el empleo del stack (pila)
—o salida	nombre del módulo de salida link-editado
—P	ejecuta sólo el preprocesador, con salida sobre el fichero de sufijo **".i"**
—p	monitoriza la ejecución
—S	produce la salida en lenguaje ensamblador sobre un fichero con sufijo **".s"**
—t[p012al]	utiliza solamente el preprocesador, compilador, ensamblador y link-editor correspondientes a los nombres de ficheros construidos con la opción **—B.**
—v	muestra el nombre de cada subproceso
—wc,arg1[,arg2...]	elimina los argumentos de paso **c** pertenecientes al conjunto **p012al**
fichero	uno o más nombres de ficheros fuentes en lenguaje C

CD — **cambia el directorio de trabajo actual**
cd [directorio]

directorio	nombre del nuevo directorio de trabajo; por defecto es el valor de **$HOME**

CHMOD — **cambia el modo de acceso a determinados ficheros**
chmod [absmode] [symmode] ficheros

abmode	4000 — inicializa el número de usuario en ejecución. 2000 — inicializa el número de grupo en ejecución. 1000 — guarda el texto después de la ejecución. 0400 — permite lectura por el propietario. 0200 — permite escritura por el propietario. 0100 — permite ejecución por el propietario. 0070 — lectura, escritura y ejecución por grupo. 0007 — lectura, escritura y ejecución por restantes usuarios.
symmode	[quien] op permiso [op permiso]
quien	
u	utiliza
g	grupo
o	otro
op	
+	añade permiso al modo de acceso al fichero
-	quita permiso
=	asignar permiso absoluto
permisos	**r** lectura **w** escritura **x** ejecución **s** inializa número identificador de propietario o grupo t guarda el texto.
ficheros	fichero(s) a cambiar

CHOWN, CHGRP — **cambia el número identificador del propietario o grupo**

chown	propietario del fichero...
chgrp	grupo del fichero...
propietario	indica el número decimal del usuario o el nombre de **login.**
grupo	indica el número decimal del grupo o un nombre de grupo correspondiente al fichero de grupos.
fichero	uno o más ficheros a cambiar al propietario o grupo especificados.

CLEAR — **borra la pantalla**
clear

CMP — **compara dos ficheros e informa de sus diferencias**
cmp [—l] [—s] fichero1 fichero2

—l	muestra el número de bytes que difieren
—s	no muestra las diferencias
fichero1	nombre de fichero
fichero2	nombre de fichero

COL — **elimina los line-feeds (saltos de línea) inversos**
col [—bfpx]

—b	suprime la posibilidad de backspace
—f	suprime los saltos de media línea
—p	muestra las secuencias de escape desconocidas encontradas a la entrada como caracteres normales, pudiendo producirse sobreimpresión debido a saltos de línea inversos
—x	suprime la conversión de espacios en blanco a tabuladores

COMM — muestra las líneas comunes a dos ficheros clasificados

comm	[—[123]] Ficherol fichero2
—1	suprime líneas sólo del ficherol
—2	suprime líneas sólo del fichero2
—3	suprime líneas de ambos ficheros
fichero1	nombre del primer fichero
fichero2	nombre del segundo fichero

CP — copia el contenido de un fichero en otro fichero o directorio

cp fichero ... objeto

fichero ... uno o más nombres de ficheros antiguos
objeto ... nuevo nombre del fichero

cp fichero ... directorio

fichero ... uno o más nombres de ficheros
directorio — nombre del directorio

CRYPT — codifica/decodificada ficheros

crypt [palabra clave]

palabra clave clave de conversión

CU — llama a otro sistema UNIX, un terminal, o un sistema no UNIX

cu [—1línea] [—svelocidad] [—t] [—h] [—d] [—m] [—o!—e] telno /dir

—1línea	específica el nombre de fichero para la línea de comunicación del dispositivo; por defecto es **/dev/ttya**
—svelocidad	especifica la velocidad de transmisión; velocidad puede ser 50, 75, 110, 134, 150, 300, 1200, 1800, 2400, 4800 ó 9600; por defecto es 300 baudios
—t	indica que la llamada es a otro terminal
—h	emula eco local, soportando llamadas a otros sistemas que requieran terminales en modo half-duplex
—d	realiza ejecución paso a paso para diagnósticos
—o	genera paridad impar para los datos enviados al terminal remoto
—e	genera paridad par para los datos enviados al terminal remoto
telno	número de teléfono de la línea
dir	indica líneas de conexión directa

DATE — muestra e inializa la fecha y hora del sistema

date [+formato] [mmddhhmm[.ss] [yy]]

yy	dos dígitos del año
mm	dos dígitos del mes
dd	dos dígitos del día
hh	dos dígitos de la hora del día
mm	dos dígitos del minuto de la hora
ss	dos dígitos del segundo del minuto

+formato formato de salida controlable por el usuario

n	inserta un carácter de salto de línea
c	inserta un carácter de tabulación
D	fecha en formato mm/dd/yy
H	hora: de 00 a 23
M	minuto: de 00 a 59
S	segundo: de 00 a 59
T	hora en formato HH:MM:SS
j	día del año: 001 a 366
w	día de la semana - Domingo=0
a	día de la semana abreviado - domingo a sábado
n	mes abreviado - enero a diciembre
r	hora en formato AM/PM

DD — realiza conversiones y copia ficheros

dd [opción=valores]...

if=ifile	especifica **ifile** como fichero de entrada; por defecto es la entrada estándard
of=ofile	especifica **ofile** como fichero de salida; por defecto es la salida estándard
ibs=n	cambia el tamaño del bloque de entrada a n bytes; por defecto es 512.
obs=n	cambia el tamaño del bloque de salida a "n" bytes; por defecto es 512
bs=n	cambia el tamaño del bloque de entrada y salida a "n" bytes; por defecto es 512
cbs=n	cambia el tamaño del buffer de conversión a "n" bytes
skip=n	salta "n" registros de entrada antes de empezar la copia
seek=n	salta "n" registros de salida antes de empezar la copia
count=n	copia sólo "n" registros de entrada
conv=tipo	donde tipo es uno de los siguientes:

ascii convierte de EBCDIC a ASCII
ebedic convierte de ASCII a EBCDIC
ibm otra conversuón de ASCII a EBCDIC
lcase convierte letras a minúsculas
ucase convierte letras a mayúsculas
swab intercambia parejas de bytes
noerror no interrumpe el proceso si hay error
sync rellena cada registro de entrada al número de caracteres especificado por **ibs** (bloqueo de entrada)
"...,..." separa los tipos de conversión mediante comas

DIFF — **muestra las diferencias entre dos ficheros**

diff [—efbn] fichero1 fichero2

—e	genera un fichero de comandos del editor para hacer fichero2 a partir de fichero1
—f	produce un texto similar al **—e** no utilizable con el editor, en orden inverso
—b	ignora los blancos posteriores
—h	produce diferencias más rápidamente
fichero1	nombre del primer fichero
fichero2	nombre del segundo fichero

DU — **muestra la cantidad de disco usado**

du [—a] [—r] [—s] [nombre...]

—a	proporciona una entrada por fichero
—r	suministra mensajes acerca de los directorios que no pueden ser leídos, abiertos, etc.
—s	muestra sólo las sumas totales
nombre	nombre del directorio o fichero

ECHO — **muestra argumentos en pantalla (para la shell)**

echo [arg]...

arg	información a mostrar en el terminal

ED, RED — **editor de textos**

ed [—] [—x] [fichero]
red [—] [—x] [fichero].

—	suprime caracteres contados por **e, r** y **w**
x	maneja ficheros encriptados
fichero	nombre del fichero a leer en el buffer del editor

Lo siguiente es un sumario de los comandos del editor en orden alfabético. El número de la línea por defecto se especifica como (.) a menos que se indique otra cosa.

(.)a	el comando **añadir** añade el texto de entrada al buffer después de la línea especificada. Introducir un punto (.) como primer y único carácter termina la inserción;
(.)c	el comando **change** cambia las líneas especificadas y acepta el texto de entrada que las reemplaza. Introducir un punto (.) como primer y único carácter para terminar los cambios;
(.,.)d	el comando **delete** cambia la línea o rango de líneas especificadas desde el buffer;
e nombre del fichero	el comando **edit** borra el contenido del buffer y copia el fichero especificado en el buffer;
E nombre del fichero	el comando **edit** cambia el contenido del buffer y copia el fichero especificado en el buffer;
f nombre del fichero	el comando **file** renombra el fichero en uso con el nombre de fichero especificado;
(1,$)g/re/command	comando **global,** interactivo marca cada línea en la que aparece la máscara /re/ y acepta el comando en cada línea marcada;
(1,$)G/re/	el comando global interactivo marca cada línea en la que aparece la máscara /re/ y acepta el comando en cada línea marcada.
h	el comando **help** da una explicación del diagnóstico **"?"** más reciente;
H	el comando **help** da una explicación del anterior y subsiguientes diagnósticos **"?";**
(.)i	el comando **insert** inserta el texto de entrada en el buffer después de la línea especificada. Introducir un punto (.) como primer y único carácter para terminar la inserción
(.,.+1)j	el comando **join** junta líneas conti-

	guas borrando el carácter de salto de línea;
(.)**kx**	el comando **mark** marca la línea especificada con el nombre "x";
(.,.)**l**	el comando **inst** imprime las líneas direccionadas y señala caracteres no-imprimibles;
(.,.)**ma**	el comando **move,** mueve la línea especificada después de la línea "a";
(.,.)**n**	el comando **number** lista el fichero señalando los números de línea;
(.,.)**p**	el comando **print** lista las líneas del texto especificadas;
P	el comando **prompt** solicita con un asterisco (*) los siguientes comandos;
q	el comando **quit** finaliza la sesión de edición y comprueba si se han producido cambios en el buffer desde la última ejecución del comando w;
Q	el comando **quit** finaliza la sesión de edición y no chequea para ver si se han producido cambios en el buffer desde la última ejecución del comando w;
($)**R** fichero	el comando **read** realiza una copia del fichero después de la línea especificada;
(.,.)**s/re/rel/**	el comando **sustituye** reemplaza la primera ocurrencia en una línea de la máscara (re) por la máscara (rel).
(.,.)**s/re/rel/g**	el comando **sustituye** realiza un cambio total de la máscara (re) por la máscara (rel) en las líneas especificadas;
(.,.)**ta**	el comando **copy** copia las líneas especificadas después de la línea "a";
u	el comando **undo** anula los efectos del comando más reciente que haya modificado el buffer;
(.,.)**v/re/rel**	reemplaza globalmente líneas en las que no se haya especificado la máscara (re);
(.,.)**V/re/**	el comando **global** interactivo marca cada línea en la que no se haya

	especificado la máscara (re) y acepta un comando para cada línea marcada;
(**1,$)w** nombre fichero	el comando **write** escribe el contenido del buffer en el fichero nombrado;
x	encripta el fichero;
($)=	número de línea de la línea mostrada en pantalla;
!comando shell	el resto de la línea a partir del carácter "!" se envía al sistema operativo interpretándose como un comando;
(.+1) **<nueva línea>**	comando de impresión que muestra línea direccionada.

ENABLE, DISABLE — **permite/impide la utilización de una impresora**

enable impresoras
disable [—C] [—R[razón]] impresoras

—c	
—r[razón]	asocia una razón con la desactivación de las improsaras;
impresoras	una o más impresoras a permitir/impedir su empleo.

ENV — **Inicializa el entorno de ejecución de un usuario (environment);**

env [—] [nombre=valor] ... [comando args]

EXPR — **Evalúa expresiones como argumentos para la shell;**
exp arg ...

arg	uno o más argumentos a evaluar;
exp	expresiones y argumentos con el orden de prioridad:

x1 /! x2	devuelve x1 si es distinto de nulo o 0, si no devuelve x2;
x1 /& x2	devuelve x1 si x1 o x2 (no ambos a la vez) son distintos de nulo o 0, si no devuelve 0.
x1 /< x2	compara si x1 es menor que x2, devolviendo 1 si es cierto y 0 si es falso;
x1 /<= x2	compara si x1 es menor o igual que x2, devolviendo 1 si es cierto y 0 si es falso;
x1 = x2	compara si x1 y x2 son iguales. Devuelve 1 si es cierto y 0 si es falso;

x1 != x2	compara si x1 es distinto de x2. Devuelve 1 si es cierto, 0 si es falso;
x1 /> x2	compara si x1 es mayor que x2. Devuelve 1 si es cierto y 0 si es falso;
x1 <= x2	compara si x1 es menor o igual que x2. Devuelve x1 si es cierto y 0 si es falso;
x1 + x2	suma x1 a x2;
x1 - x2	resta x2 a x1;
x1 * x2	multiplica x1 por x2;
x1 / x2	divide x1 por x2;
x1 % x2	devuelve el valor de x1 resto módulo x2.

FILE — **determina el tipo de un fichero.**
fichero [—c] [—f ffile] **[—m** mfile] ficheros

—f ffile	fichero conteniendo los nombres de ficheros a examinar;
—m mfile	utiliza **mfile** como "número mágico" (indicador del tipo de fichero);
—c	comprueba el "número mágico" para detectar errores de formato;
ficheros	uno o más ficheros a examinar.

FIND — **Busca ficheros.**
find pathname - list - expresión.

panthname-list	uno o más nombres de ficheros;
expresión	busca el modelo indicado, donde **n** es un entero, **+n** es mayor que **n** y **—n** es menor que **n.**

Expresiones permitidas:

—atime n	cierto si el fichero ha sido accedio en los últimos **n** días;
—ctime n	cierto si el fichero ha sido cambiado en los últimos **n** días;
—exec comando	cierto si la ejecución del **comando** devuelve un 0;
(expresión)	cierto si la expresión entre paréntesis "()" es cierta;
—group gname	cierto si el fichero pertenece al grupo **gname;**
—links n	cierto si el fichero posee "n" enlaces (links);
—mtine n	cierto si el fichero ha sido modificado en **"n"** días;
—name nombre fichero	cierto si el nombre de fichero coincide con el dado;
—ok comando	lo mismo que el comando **exec,** pero se ejecuta sólo tras recibir conformidad por parte del usuario;
—newer nombre fichero	cierto si el nombre de fichero fue modificado antes que el fichero;
—perm onum	cierto si coinciden los indicadores de modo de acceso en octal;
—print	muestra el nombre completo del fichero;
—size n	cierto si el fichero ocupa **"n"** bloques de longitud;
—type x	cierto si el fichero tiene el tipo **"x",** donde **"x"** puede ser una **"b"** para un fichero especial organizado en modo bloques, **"c"** para un fichero especial organizado en modo carácter, **"d"** para un directorio y **"f"** para un fichero normal;
—user nombre	cierto si el fichero pertenece al usuario.

GREP, EGREP, FGREP — **Búsqueda de cadenas y expresiones.**

grep	[—v] [—c] [—l] [—n] [—b] [—s] limexpr [ficheros];
egrep	[—v] [—c] [—l] [—n] [—b] [—e expresión] [—f fichero] [regexpr] [ficheros];
fgrep	[—v] [—x] [—c] [—l] [—n] [—b] [—e expresión] [—f ficheros] [strings] [ficheros]

—v	imprime todas las líneas excepto las que son iguales;
—x	imprime sólo las líneas que son completamente iguales (fgrep);
—c	imprime un contador de las líneas iguales;
—l	lista nombres de ficheros con líneas iguales;
—n	cada línea precedida por el número de línea;
—b	cada línea precedida por el número de bloque que fue construido;
—s	imprime mensajes de error para ficheros no existentes (grep);
—e expresión	permite a las expresiones empezar con — (egrep, fgrep);
—f fichero	coge la expresión regular (egrep) o lista de strings (fgrep) del fichero;

limexpr	busca la expresión especificada; debe limitarse a expresiones regulares de la forma de ed;
regexpr	busca una expresión regular;
strings	busca un string;
ficheros	para buscar en uno o más ficheros.

HEAD — **muestra las primeras líneas de un fichero**
head [—contador] [fichero...]

—contador	proporciona un número especificado de líneas;
fichero...	uno o más nombres de ficheros.

KILL — **finaliza un proceso**
kill [—sig] identificador-proceso...

—sig	1 — entrada en bucle (SIGNUP);
	2 — interrupción (SIGINT);
	3 — salir (SIGQUIT);
	4 — instrucción ilegal (SIGILL);
	5 — trace trap (SIGTRAP);
	6 — instrucción IOT (SIGIOT);
	7 — instrucción EMT (SIGEMT);
	8 — error en operación en coma flotante (SIGFPE);
	9 — finalizar (SIGBUS);
	10 — error de bus (SIGBUS);
	11 —. violación de segme ntación de memoria (SIGSEGV);
	12 — error en llamada al núcleo (system call) (SIGSYS);
	13 — escribir en un pipe sin ningún proceso en lectura (SIGPIP);
	14 — alarma por temporización (SIGALARM);
	15 — terminación software (SIGTERM);
	16 — no asignado.
id. de proceso	uno o más números de identificación de procesos.

LINT — **chequeo de programas en C**
lint [—abhlnpuvx] fichero...

—a	suprime errores de asignaciones de valores long a variables enteras;
—b	suprime errores de sentencias no alcanzadas;
—h	no intenta encontrar errores de programación, ni estilo no apropiado y reduce comentarios;
—lx	incluye la librería lint llib-lix.ln;
—n	no chequea compatibilidad con la librería estándard;
—p	chequea portabilidad a IBM y GCOS;
—u	suprime mensajes de funciones y variables sin utilizar;
—v	suprime mensajes de argumentos de funciones sin utilizar
—x	suprime mensajes de variables externasx nunca utilizadas;
fichero...	uno o más nombres de ficheros.

LOGIN — **conexión al sistema**
login [nombre de usuario [env-var ...]].

nombre de usuario	cambia un usuario ya conectado al sistema por otro usuario;
env-var	argumentos para expandir o modificar el entorno de ejecución (environment).

LP — **gestiona el manejo de una impresora**
lp [—c] [—d dest] [—m] [—n número] [—o opción] [—s] [—t título] [—w] ficheros.

—c	hace copias de los ficheros a imprimir inmediatamente antes de la llamada;
—d dest	envía petición a dest;
—m	envía mensaje después de que se imprimen los ficheros;
—n número	número de copias; por defecto es 1;
—o opción	especifica las opciones que dependen de la impresora;
—s	suprime los mensaje de lp;
—t título	imprime el título en la cabecera de página;
—w	escribe un mensaje en el terminal después de imprimir los ficheros;
ficheros	uno o más ficheros para ser imprimidos.

LS — **lista el contenido de un directorio**
ls [—AaCcdFfgiLlqRrstu1].

—A	lista todas las entradas excepto "." y "..";
—a	lista todos los ficheros;
—C	fuerza la salida en multi-columna;

—c usa la flecha de creación del fichero para su clasificación (—t) o impresión (—l);
—d sólo lista el nombre si el argumento es un directorio;
—F marca los directorios con una barra "/", los ficheros ejecutables con un "*", los pipes con "=" y los links simbólicos (enlaces) con "@";
—f fuerza que cada argumento sea interpretado como un directorio;
—g incluye el número identificador del grupo en formato largo;
—i escribe el número de i-noda en la primera columna de cada fichero;
—L lista el fichero o enlaces de referencias a directorios si su argumento es un link (enlace) simbólico;
—l lista en formato largo;
—q fuerza la impresión de los caracteres no gráficos en los nombres del fichero;
—R lista subdirectorios;
—r clasifica en orden inverso;
—s da el tamaño en bloques;
—t ordena por la fecha de modificación;
—u clasifica por la fecha de último acceso;
—1 fuerza un fichero por línea.
nombre ... uno o más nombres de directorios de ficheros.

M4 — **procesador de macros en texto**

m4 [—Bint] [—e] [Hint] [—Sint] [—s] [—Tint] [—D nombre] [=vall] [—U nombre] [ficheros]

—Bint Cambia el tamaño de la pila y buffers de argumentos; por defecto 4096 bytes;
—e opera interactivamente;
—Hint cambia el tamaño de la tabla de símbolos; por defecto 199 bytes;
—Sint cambia el tamaño del stack; por defecto 100 slots;
—s permite líneas sync
—Tint cambia el tamaño del buffer de comandos; por defecto 512 bytes;
—D nombre [=vall] define el nombre como "val" o el valor nulo si no se especifica "val";
—U nombre no define nombres;
ficheros uno o más nombres de ficheros a procesar.

MAIL, RMAIL, SMAIL — **envía/lee mensajes a/de usuarios**

mail [—t] usuarios
mail [—f ficheros] [—epqr]
rmail [—t] usuarios
smail [—t] usuarios

usuarios... usuario(s) a quienes se envía el mensaje
—f ficheros envía mensaje a *fichero* en lugar de al fichero mail (defecto);
—e no imprime mensajes; devuelve 0 si el usuario tiene mensajes o 1 si no los tiene;
—p imprime el fichero de mensajes;
—q produce la terminación de mail tras una interrupción;
—r orden "primero en entrar primero en salir" (FIFO);
—t pone en el buzón los nombres de todas las personas a las cuales se envió el mensaje.

Comandos interactivos

d	borra mensajes y continúa con el siguiente
EOT (control-D)	cierra el fichero de mensajes y termina
m[usuarios]...	envía el mensaje a uno o más usuarios
tecla newline	continúa con el siguiente mensaje
P	imprime el mensaje de nuevo
q	cierra el fichero de mensajes y termina
s[fichero]	salva el mensaje en el fichero nombrado
X	sale sin cambiar el fichero de mensajes
w[fichero]	salva el mensaje en el fichero nombrado sin cabeceras
!comando	ejecuta un comando de la shell
?	imprime el resumen de los comandos
—	vuelve al mensaje anterior

MKDIR — **crea un directorio**

mkdir dirname

dirname... uno o más nombres de directorios

MORE, PAGE — **filtro de salida a pantalla para realizar paginación de ficheros**

more [—cdfslu] [—n] [+número de línea!+/pattern] page **[nombre...]** [—cdfslu]

—c borra cada línea antes de sobreimprimirla
—d muestra el mensaje "pulsar espacio para continuar" ("Hit return to continue")

—f	no trunca las líneas de longitud excesiva
—s	reduce múltiples líneas en blanco a una sola
—l	no trata los ^L (form feed) de forma especial
—u	no permite el subrayado en el terminal
—n	especifica el número de las líneas a representar
+número de línea	empieza por el número indicado
+/pattern	empieza dos líneas después de la que contiene la máscara
nombre...	uno o más ficheros a visualizar

Comandos interactivos:

[I]<espacio>	muestra "i" líneas adicionales u otra pantalla si no se especifica argumento
ôD	muestra 11 líneas más
v	empieza el editor vi en la línea en curso
h	ayuda
,	vuelve al punto en el que se inició la última búsqueda
i/expr	busca la ocurrencia iésima de la expresión (expr)
if	salta "i" pantallas
is	salta "i" líneas
iz	especifica una nueva ventana de tamaño "i"
q	sale del comando more
!cmd	invoca al comando shell
:f	muestra el nombre y el número de línea del fichero en curso
	repite el comando anterior

NICE — **modifica la prioridad de proceso**

nice [—número] comando [argumentos]

—número	número de prioridad; la mayor prioridad es 0 y la menor 19
comando	comando a ejecutar con la prioridad más baja
argumentos	argumentos para el comando especificado

NROFF — **formatea ficheros de texto**

nroff [—c nombre] [—e] [—h] [—i] [—k nombre] [—m nombre] [—nN] [—o lista] [—q] [—raN] [—sN] [—T nombre] [un] [—z] [fichero]

—c nombre	utiliza ficheros de macros usr/lib/macros/cmp.[nt].[dt].nombre y usr/lib/macros/ucmp.[nt].nombre
—e	separa las palabras ajustando las líneas
—h	usa los tabuladores horizontales
—i	lee las entradas estándar después que los ficheros
—k nombre	coloca los ficheros de macros computados en [dt].nombre
—m nombre	prepara el fichero de macros /usr/lib/tmac/tmac.nombre
—nN	usa N como primera página
—o lista	lista las páginas especificadas en la salida o separadas por un rango (N-M)
—q	realiza entrada/salida simultánea ante un comando "rd"
—raN	inicializa el registro a N
—sN	detiene el listado cada N páginas
—T nombre	envía la salida al nombre del terminal especificado
—un	inicializa el número de sobreimpresiones (en negrita) a "n"
—z	imprime sólo los mensajes generados por la petición ".tm"
fichero	fichero para ser formateado

OD — **volcado en octal**

od [—bcdosx] [fichero] [[+] offset [.] [b]]

—b	interpreta los bytes en octal
—c	interpreta los bytes en ASCII
—d	interpreta palabras en decimal sin signo
—o	interpreta palabras en octal
—s	interpreta palabras de 16 bits en decimal con signo
—x	interpreta palabras en hexadecimal
fichero	fichero a volcar
+	especifica si se omite el nombre de fichero anterior
offset	especifica el desplazamiento (offset) inicial
.	interpreta el desplazamiento en decimal
b	interpreta el desplazamiento en bloques de 512 bytes

PASSWD — **cambia la clave de acceso**

passwd	[nombre]
nombre	nombre del usuario

PR — **imprime ficheros**

pr [—a] [—d] [—eck] [—f] [—h] [—iick] [—ln] [—m] [—n] [+n] [—nck] [—ok] [—p] [—r] [—sc] [—t] [—wn] [fichero ...]

—a	imprime en varias columnas a lo ancho de la página
—d	salida a doble-espacio
—eck	expande los tabuladores a su entrada como espacios en blanco "c" se retiene al carácter de tabulación de salida
—f	lista con saltos de línea
—h hd	usa hd como cabecera en lugar del nombre del fichero
—ick	reemplaza los espacios en blanco por tabuladores
—ln	usa "n" como longitud de la página, por defecto es 66
—m	imprime todos los ficheros en columnas separadas
—n	salida en "n" columnas
+n	empieza en la página "n"
—nck	utiliza una anchura de "k" dígitos para numeración de líneas (por defecto "k" es 5)
—ok	desplaza cada línea en "k" caracteres
—p	realiza una pausa antes de imprimir cada página
—r	no imprime errores de apertura de ficheros
—sc	separa las columnas con el carácter "c"
—t	no imprime las 5 líneas de cabecera o pie de página
—wn	ancho de página de "n" caracteres, por defecto es 72
fichero...	uno o más nombres de ficheros

PRINT/LPR — **envía ficheros al spooler para imprimirlos**

print ! lpr [—b] [—cp nnn] [—fm xxxxx] [—ln nn] [—pr nn] [—pt lpnn] ficheros

—b	imprime con cabecera (banner)
—cp nnn	especifica el número de copias (nnn) a imprimir, por defecto 1
—fm xxxxx	designa la cola de impresión correspondiente al tipo de formulario (xxxxx)
—ln nn	especifica el número de líneas por pulgada (nn), por defecto 6
—pr nn	especifica el número de prioridad (nn) para la impresión
—pt lpnn	reencamina la salida a la impresora especificada (lpnn)
ficheros	uno o más nombres de ficheros a imprimir

PRINTENV — **imprime las variables del entorno de ejecución (environment)**

printenv

PS — **lista información sobre procesos activos**

ps [—a] [—d] [—e] [—f] [—g GLIST] [—l] [—n nombre del listado] [—p plist] [—s swapdev] [—t tlist] [—u ulist]

—a	lista información sobre todos los procesos asociados con el terminal
—d	lista la información de todos los procesos, excepto los procesos del grupo
—e	lista información de todos los procesos
—f	genera un listado completo
—g glist	lista sólo los procesos del grupo especificados en glist
—l	genera un listado en formato largo
—n lista de nombres	usa lista de nombres como alternativa
—p plist	lista sólo los procesos especificados
—s swapdev	usa swapdev en lugar de /dev/swap/
—t tlist	lista sólo los procesos asociados con los terminales especificados en tlist.
—u ulist	lista sólo los procesos asociados con los usuarios especificados en ulist.

RM, RMDIR — **borra ficheros o directorios**

rm [—f] [—i] [—r] fichero...

rmdir fichero...

—f	fuerza separaciones
—i	pregunta interactiva antes de borrarlos
—r	borra un directorio y todos los ficheros del mismo
fichero	uno o más nombres de ficheros o directorios

SH — **lenguaje de programación de comandos**

sh [—ceiknrstuvx] [argumento]......

—c string	usa string como un comando
—e	un mal status de salida si no interactivo
—i	entra modo interactivo
—k	pone palabras clave en un entorno para un comando
—n	lee, pero no ejecuta comandos
—r	restringe el entorno
—s	lee desde la entrada estándar
—t	sale después de ejecutar un comando

—u	trata las variables no agrupadas como un error
—v	imprime las líneas de entrada cuando son leídas
—x	imprime el comando cuando ejecuta
arg...	uno o varios argumentos

Caracteres especiales de la shell

;	ejecuta secuencialmente el procedimiento pipeline
@	ejecuta sin esperar a terminar el comando
@	ejecuta el comando si el comando anterior devolvió 0
'...'	cita todos los caracteres que están limitados entre las comillas
/	cita todos los caracteres que se encuentran detrás de la barra invertida
"..."	comandos y parámetros a sustituir
#text	considera todo el texto introducido hasta el final de la línea como comentario

Generación de un nombre de fichero de la shell

*	cualquier string de caracteres
?	cualquier carácter
[..]	cualquiera de los caracteres encerrados

Sentencias de comandos de la shell

case palabra in [pattern]
[... list;;] ...esac
for name [in palabra..] do lista done
if lista then lista [elif lista then lista]...[else lista] fi
(lista)
{lista}
while lista [do lista] done

Parámetros de sustitución de la shell

$#	número de parámetros
$—	flag suministrado por invocación
$?	valor retornado del último comando ejecutado
$$	número de proceso de la shell
$!	número del último comando (background)
$HOME	especifica el directorio home
$IFS	especifica separadores de campo interno
$MAIL	especifica el fichero mail
$PATH	especifica el fichero PATH
$CDPATH	especifica la búsqueda
$PS1	especifica la prompt primaria, $ por defecto
$PS2	especifica la prompt secundaria, > por defecto
$TERM	especifica tipo de terminal
nombre=valor	especifica valores de parámetros
$ {parámetro}	utiliza un conjunto de parámetros
$ {parámetro:? palabra}	utiliza parámetros si está agrupado y es no nulo, si no imprime palabra y después sale
$ {parámetro:+ palabra}	utiliza palabra si el parámetro está agrupado o es nulo, en otro caso nada
<palabra	la palabra del fichero como entrada estándar
<<palabra [—]	utiliza la palabra del fichero como entrada estándar seguido por identificación de línea, palabra o fin de fichero
>palabra	utiliza la palabra del fichero como salida estándar
>>palabra	utiliza la palabra del fichero como salida estándar, lo añade a la salida si el fichero ya existe
>fdígito	duplica la entrada estándar para el dígito del descriptor de fichero
<f—	cierra la entrada estándar

Comandos especiales de la SH

.fichero	lee y ejecuta comando del fichero y vuelve
break [n]	sale de un for o while en "n" niveles
cd[arg]	cambia el directorio actual a arg; directorio home por defecto
continue[n]	resume n-iteración de un for o while
eval[arg.]	lee y ejecuta argumentos
exec[arg.]	ejecuta argumentos en lugar de la shell
exist[n]	sale con el status "n"
export[name.]	exporta el nombre para un entorno de comandos

login[argI] — cambia de usuario
newg.p[arg...] — cambia de grupo
read name... — lee de la entrada estándar
readonly [nombre...] — marca el nombre como sólo lectura
set[—ekntuvx[arg...]]

—e	status mal si no interactivo
—k	pone palabra clave en un entorno para un comando
—n	lee pero no ejecuta comandos
—t	sale después de que se ejecute un comando
—u	trata las variables no agrupadas como un error
—v	imprime líneas de entrada cuando son leídas
—x	imprime comandos cuando lo ejecuta
——	no cambia cualquiera de las opciones
arg...	uno o más argumentos
shift	renombra argumentos
test	evalúa expresiones condicionales
times	imprime tiempos de procesos acumulados

trap [arg][n]

—f	impone como tamaño límite "n" bloques en los ficheros de los procesos hijos
—p	cambia el tamaño del pipe a "n"
unmask [nnn]	agrupa la máscara del fichero al valor nnn
wait [n]	espera por el número [n] de identificación especificado e imprime el status de la terminación
file	lee y ejecuta comandos del fichero

SIZE — **lista el tamaño de un fichero objeto**

size [—x] [—o] [—v] [objeto...]

—x	imprime números en hexadecimal
—o	imprime números en octal
—v	imprime la información de la versión
objeto	uno o varios nombres de ficheros objetos

SLEEP — **suspende la ejecución por un intervalo de tiempo**

sleep time

time	suspende por el número especificado en segundos

SORT — **ordena o mezcla ficheros**

sort [—cmu] [—o nombre] [—dfinr] [—btx] [+posición] [—posición2] [ficheros]

—b	ignora blancos repetitivos
—c	chequea que los ficheros de entrada estén clasificados de acuerdo a las reglas
—d	clasifica en orden de diccionario
—f	igual tratamientos mayúsculas que minúsculas
—i	ignora caracteres fuera de código ASCII desde el 40 hasta el 176
—m	mezcla ficheros (merge)
—n	clasifica en orden aritmético
—r	clasifica en orden inverso
—u	saca una copia de las líneas iguales
+pos1[—pos2]	la clave de clasificación comienza en la posición 1 y finaliza en la 2
—o nombre	usa 'nombre' como fichero de salida
ficheros...	uno o varios ficheros

SU — **entrada en superusuario**
su [—] [nombre] [argumento]

—	cambia los parámetros del usuario como si se estuviese entrando en uno nuevo
nombre	nombre de usuario
argumentos	argumentos para la shell

SYNC — **actualiza periódicamente el súper-bloque**
sync

TAIL — **lista la última parte de un fichero**
tail [+—[contado][lbc]] [fichero]

+contador	número de líneas desde comienzo de fichero
—contador	número de líneas desde fin de fichero
b	contador de bloques
c	contador de caracteres
l	contador de líneas
fichero	nombre del fichero

TAR — **efectúa copias en formato de archivo (archive)**
tar [clave] [nombres de fichero]

clave	controla la acción del comando; las claves pueden ser:
0... 7	selecciona el drive
c	crea una nueva copia
l	visualiza mensajes de error si existe algún link incorrecto

m	no recupera ficheros que se hayan modificado posteriormente
t	lista los nombres de los ficheros
v	lista nombres de fichero seguidos por más información
w	espera confirmación de cada fichero a copiar o recuperar

TEST — **evalúa la condición de un comando**

test [—b fichero] [—c fichero] [—d fichero] [—f fichero] [—g fichero] [—k fichero] [—ns1] [n1 —eqn2] [—p fichero] [—r fichero] [—s fichero] [s1] [s1=s2] [s1 != s2] [—t fildes] [—u fichero] [—w fichero] [—x fichero] [—z s1]

—b fichero	verdadero si el fichero existe y es un bloque especial de fichero
—c fichero	verdadero si el fichero existe y si es un carácter especial de fichero
—d fichero	verdadero si el fichero existe y es un directorio
—f fichero	verdadero si el fichero existe y es regular
—g fichero	verdadero si el fichero existe y su bit de set-grupo ID es cambiado
—k fichero	verdadero si el fichero existe y su bit de sticky es cambiado
—ns1	verdadero si la longitud del strint (s1) no es 0
n1 —eqn2	verdadero si los enteros n1 y n2 son iguales
—p fichero	verdadero si el fichero existe y es llamado pipe
—r fichero	verdadero si el fichero existe y es legible
—s fichero	verdadero si el fichero existe y su tamaño es mayor que 0
s1	verdadero si la cadena s1 no es nula
s1=s2	verdadero si las cadenas s1 y s2 son iguales
s1!=s2	verdadero si las cadenas s1 y s2 no son iguales
—t fildes	verdadero si el número descriptor del fichero está asociado con el terminal, por defecto es 1
—u fichero	verdadero si el fichero es 1 y su bit de set-usuario ID es cambiado

—w fichero	verdadero si el fichero existe y se puede escribir
—x fichero	verdadero si el fichero existe y es ejecutable
—z s1	verdadero si la longitud de la cadena es 0

TIME — **tiempo de un comando**

time comando

commando	comando a ser cronometrado

TRUE — **provee los valores de verdad**

true
false

TTY — **lista el nombre completo de un terminal**

tty [—l] [—s]

—s	suprime la impresión del nombre del terminal; permite ver el código de salida

UMASK — **cambia la máscara del modo de creación de un fichero**

umask [ooo]

ooo	cambia la máscara del modo de creación de un fichero a ooo.

VI — **editor orientado a pantalla**

vi [—t tag] [—r] [+comando] [—l] [—wn] nombre...

—t tag	equivalente a un comando inicial tag
—r	recobra la última versión salvada del nombre del fichero especificado
+comando	empieza la ejecución del comando especificado
—l	cambia las opciones showmatch y lisp de LISP
—wn	cambia el tamaño por defecto de la ventana a "n"
nombre...	uno o más nombres de ficheros

Lo siguiente es una lista de los comandos del editor VI. Cada comando de la lista está agrupado de acuerdo con su función de edición. Los comandos Ex pueden ser ejecutados en el editor VI precediendo el comando con a:. Todos los comandos con la palabra CONTROL son mantenidos por la llave CONTROL y preceden a la llave indicada.

Comandos de cancelación del VI:

Q	Termina el modo de editor VI
—ZZ	Escribe el buffer en el fichero actual y sale del editor

Comandos de movimiento del cursor:

CONTROL-B or b	mueve el cursor hacia atrás hasta el principio de una palabra
CONTROL-B	vuelve a la página anterior
CONTROL-D	baja la pantalla media ventana
CONTROL-E	escribe una línea adicional al final de la pantalla en curso
CONTROL-F	siguiente pantalla
CONTROL-G	visualiza, si el fichero se ha modificado, el número de la línea en curso, número de líneas en el fichero y porcentaje de localización
CONTROL-H or h	mueve el cursor una posición hacia atrás
CONTROL-J	avanza a la siguiente línea, en la misma columna
CONTROL-M	avanza a la siguiente línea
CONTROL-N or j	la siguiente línea en la misma columna
CONTROL-P or k	a la línea anterior en la misma columna
CONTROL-U	mueve la pantalla hacia arriba media ventana
CONTROL-Y	visualiza una línea adicional al principio de la pantalla en curso
—B	mueve el cursor hasta el principio de una palabra
E or e	mueve el cursor hasta el final de la palabra en curso
Fx	localiza o encuentra el primer carácter "x"; no modifica la línea en curso
fx	localiza o encuentra el primer carácter "x" desde la línea en curso
G	va al número de línea especificado
H	mueve el cursor a la primera línea
L	mueve el cursor al primer carácter de la primera línea
M	mueve el cursor a la línea central de la pantalla
N	invierte el orden de búsqueda
n	repite la última búsqueda
Tx	localiza el carácter "x" después del cursor y lo coloca después del carácter

tx	avanza el cursor hacia arriba, hasta el carácter detrás de "x"
W	mueve el cursor hasta el principio de la siguiente palabra en la línea
w	mueve el cursor al principio de la siguiente palabra
SPACE or l	avanza el cursor hasta el siguiente carácter en la línea
+	avanza el cursor al primer carácter que no sea blanco de la línea siguiente
—	posiciona el cursor en el primer carácter que no sea blanco en la línea anterior
/string	examina desde la siguiente ocurrencia en la cadena
?string	examina hasta la siguiente ocurrencia en la cadena
i	mueve el cursor hasta la posición del primer carácter no blanco
;	repite el último carácter encontrado de f, F, t y T
$	mueve el cursor hasta el final de la línea en curso
%	mueve la posición del cursor a los paréntesis o llaves cuya máscara está en curso
)	línea siguiente
(	línea anterior
}	párrafo siguiente
{	párrafo anterior
]]	sección siguiente
[[	sección anterior

Comandos de manipulación de texto del vi:

A	añade al final de la línea
a	añade después de la posición del cursor
C	cambia el texto de la línea en curso por otro
CONTROL-?	interrumpe el editor para volver al estado de entrada del comando
D	borra el texto desde la posición del cursor hasta el final de la línea
d obj	borra el objeto especificado
ESC	llave de escape para terminar una entrada de A, a, C, c, S y s
I	inserta al principio de la línea
i	inserta después de la posición del cursor.
J	junta líneas

O inserta una línea por encima de la línea en curso

o inserta una línea por debajo de la línea en curso

P o p recupera el último texto borrado después/antes de la posición del cursor

R reemplaza caracteres

rc reemplaza el carácter en el que está posicionado el cursor por c

S sustituye la línea por las líneas introducidas

s sustituye el carácter en el que está el cursor por una cadena de caracteres

U deja la línea en curso como estaba antes de hacer cambios

u recupera el último cambio hecho en la línea

X borra el carácter anterior al cursor

x borra el carácter en el que está el cursor

ny copia "n" líneas en un buffer, lo recupera con P/p

. repite el último cambio

Comandos de ajuste de la pantalla

CONTROL-L limpia y vuelve a dibujar la pantalla

CONTROL-R vuelve a dibujar la pantalla en curso, eliminando las líneas

Comandos diversos del vi

CONTROL-I cuando se inserta imprime un número o espacio como el hecho por la opción tabstop

CONTROL-Q o V utiliza inserciones de no-impresión y caracteres especiales

CONTROL-S suspende la salida hasta que se introduzca otro CONTROL-S

CONTROL-W mueve el cursor hasta el principio de la palabra anterior

CONTROL-[cancela un comando con forma parcial

CONTROL-] busca por un tag a partir de la posición del cursor

CONTROL-| vuelve a la posición previa en el último fichero editado

CONTROL-@ reemplaza el último texto insertado

m letra marca la posición en curso con letra

0 se mueve hasta el primer carácter de la línea en curso

1-9 forma argumentos numéricos

: prefija el comando ex y manipula la opción

< cambios a la izquierda

> cambios a la derecha

= cambia el indentado de la línea del LISP

" vuelve al contexto anterior

!obj cmd procesa el objeto especificado en el buffer

"nombre precedido de un nombre de buffer; el número/nombre puede ser desde 1 hasta 9 para salvar el texto borrado y desde a hasta z para textos almacenados

#número sustituye el número por una función en el terminal

@ repite la sustitución anterior

vi opciones de manipulación

autoindent suministra indentación automáticamente; por defecto es noai

ignorecase ignora caso de búsqueda; por defecto es noic

lisp comandos ({) } i; por defecto es nolisp

list el tabulador imprime como I, fin de línea marcado como $; por defecto es nolist

magic los caracteres ".", "[" y "*" son especiales en las exploraciones; por defecto es nomagic

number las líneas son imprimidas con un número de líneas prefijadas; por defecto es nonu

paragraphs con los macros nombrados empieza a dividir los párrafos; por defecto es para = IPLPPPQPbpbPLI

redraw simula un terminal inteligente; por defecto es nore

sections para empezar nuevas secciones; por defecto es = NHSHH HU

shiftwidth cambia distancia para <and>; por defecto es SW = 8

showmatch muestra acompañado de (or {as)or } ; por defecto es nosm

slowopen pospone mostrar fuera de fecha mientras inserta; por defecto es slow

term especifica el tipo de terminal que se utiliza

wrapscan busca el fin del buffer si no se encuentra ningún string; por defecto es ws

WC — **cuenta líneas, palabras y caracteres en un fichero**

wc [—lwc] [fichero ...]

—c cuenta sólo caracteres

—l	cuenta sólo líneas
—w	cuenta sólo palabras
fichero...	uno o más nombres de ficheros

WHO — **lista todos los usuarios en el sistema**
who [—uTlpdbrtas] [fichero] [am I]

—u	lista información acerca de los usuarios que estén en el sistema
—T	si alguien puede o no escribir a ese terminal; + si puede, — en caso contrario
—l	lista la línea en la que el sistema está esperando para que alguien entre con login
—p	lista los procesos activos actualmente previamente producidos por el init
—d	imprime los procesos terminados no producidos por el init
—b	muestra fecha y hora de la última inicialización
—r	indica el nivel de ejecución actual del proceso init
—t	indica el último cambio de root al reloj del sistema
—s	lista sólo el nombre, línea, hora
fichero	examina el fichero especificando en lugar del letc/utmp
am i	lista el nombre del usuario

WHOAMI — **lista el nombre del usuario**
whoami

WRITE — **Envía mensajes a otro usuario**
write usuario [ttyname]

user	nombre del usuario
ttyname	nombre del terminal (si el nombre de usuario está en más de un terminal)

BIBLIOGRAFIA

"UNIX, sistema y entorno".
Fontaine y Hammes. *Ed. Massón.* Barcelona, 1986.

"Unix. Guía Profesional".
Banaham, M. F. *Ed. Díaz de Santos.* Madrid, 1986.

"El sistema UNIX y sus aplicaciones".
Canosa, José. *Ed. Marcombo. Boixareu Editores.* Barcelona, 1984.

"Sistema Operativo Unix: Guía del Usuario".
Thomas Rebeccas y Yates, Jean. *Osborne McGraw-Hill.* Berkeley, California, 1984.

"Unix Programmer's Manual".
Bell Telephone Laboratories. *Bell Telephone Laboratories, Murray Hill,* New Jersey, 1979.

"Making use of Unix".
Budgen, D., 1984.

"Real world UNIX. Managing a business with the UNIX operating system".
Halanka, J. D., 1984.

"UNIX. Mecanismes de base. Langage de comande. Utilisation".
Lucas-Martin y De Sablet. *Ed. Eyrolles.* París, 1984.

"The Unix Operating System".
Ritchie, Dennis M., S. C. Johnson, M. E. Lesk y Brian W. Kernigham. *The Bell System Technical Journal (BSTJ).* Vol. 57, número 6, julio-agosto, 1978.

"The UNIX System Guide Book. An introductory guide for serious users".
Silvester, P. P., 1984.

NOTAS



Desde su aparición comercial a finales de los 70, el sistema operativo UNIX se ha convertido en un estándar de mercado para los modernos microordenadores multiusuario y multitarea de 16 y 32 bits. El número de fabricantes y usuarios que se sirven de él en los más diversos campos de aplicación va en constante aumento. Este volumen de la Biblioteca Básica Informática busca ser una primera introducción al UNIX, describiendo sus orígenes, estructura, elementos y posibilidades de una manera sencilla, explicando claramente los conceptos básicos y facilitando al lector su primer contacto con UNIX y C, de forma que se encuentre en condiciones, si lo desea, de acceder a obras más especializadas.

395 pts.
(incluido IVA)